ENCYCLOPEDIA FOR THE TRS-80*

A library of useful information for your TRS-80

Business
Education
Games
Graphics
Hardware
Home Applications
Interface
Tutorial
Utility

VOLUME 9

*Trademark of Tandy Corp.

ENCYCLOPEDIA
for the TRS-80*

ENCYCLOPEDIA for the TRS-80*

VOLUME 9

wayne GREEN inc.
PETERBOROUGH NH 03458

*Trademarks of Radio Shack Division of Tandy Corp.

The LDOS disk operating system, a product of Logical Systems, was used in the technical production of this book.

FIRST EDITION
FIRST PRINTING JULY 1982

Edited by Kate Comiskey and Katherine Lindquist
Proofread by Ann Winsor
Production: Margaret Baker, Gary Ciocci,
Linda Drew, Thomas Villeneuve, Robert M. Villeneuve,
John R. Schweigert, Sandra Dukette, Elizabeth Libby, Karen Stewart
Technical Editor: Jim Heid
Illustrations by Howard Happ

FOREWORD

The Biggest Difference

There are lots of arguments about which computer is the best. The answer to this question lies not in which hardware is best. That is really irrelevant, when you understand the field. The major value of any computer lies in the software and the information available for it. Hence this encyclopedia.

The TRS-80 is by no means the best computer on the market as far as its hardware is concerned, but with the support of *80 Microcomputing* magazine and this encyclopedia series, you have an almost unlimited source of information on how to use your computer—and of programs. With this information source the TRS-80 is by far the most valuable computer system ever built. No other computer, at any price, has anything approaching this amount of user information and programs available.

Most encyclopedias try to freeze everything at one time and are thus able to divide the material up alphabetically. This is a new kind of encyclopedia —a living one—with each new volume keeping you up to date on the very latest information on using your computer and the newest of programs.

Your computer can be a fantastic teaching device, a simulator, a way to play all sorts of fascinating games, a business aid, a scientific instrument, a control unit for machinery. . . . It is one of the most flexible gadgets ever invented. All of these applications are possible *if* you have the information and the programs. This encyclopedia will give you these.

To get the best use of your TRS-80, don't miss a single volume of the *Encyclopedia for the TRS-80.*

WAYNE GREEN
Publisher

CONTENTS

Please note: Before typing in any listing in this book, see Appendix A.

Encyclopedia Loader™

The editors of Wayne Green Books want to help you maximize your microcomputing time, so they created the **Encyclopedia Loader™**.

The **Encyclopedia Loader** is a special series of cassettes that offer the longer programs in the **Encyclopedia for the TRS-80*** in ready-to-load form. Each of the ten volumes of the Encyclopedia provides the essential documentation for the programs on the Loader.

With the **Encyclopedia Loader**, you'll save hours of keyboard time and eliminate the aggravating search for typos. The **Encyclopedia Loader** for Volume 9 will contain the programs for the following articles:

Layaway
Your Fair Share
Do-It-Yourself Maze Package
Munch
Autocost
Your Personal Expense Account
TRSDOS Multiple Command Processor
Dandyzap

Encyclopedia Loader™ for Volume 1	**EL8001**	**$14.95**
Encyclopedia Loader™ for Volume 2	**EL8002**	**$14.95**
Encyclopedia Loader™ for Volume 3	**EL8003**	**$14.95**
Encyclopedia Loader™ for Volume 4	**EL8004**	**$14.95**
Encyclopedia Loader™ for Volume 5	**EL8005**	**$14.95**
Encyclopedia Loader™ for Volume 6	**EL8006**	**$14.95**
Encyclopedia Loader™ for Volume 7	**EL8007**	**$14.95**
Encyclopedia Loader™ for Volume 8	**EL8008**	**$14.95**
Encyclopedia Loader™ for Volume 9	**EL8009**	**$14.95**

(Please add $1 50 per package for postage & handling)

Mail your order to "Encyclopedia Loader Sales," Wayne Green Books, Pine Street, Peterborough, NH 03458 or call **(1-800-258-5473)**.

*TRS-80 is a trademark of Radio Shack Division of Tandy Corp.

BUSINESS

Layaway
Your Fair Share

Layaway

Miguel Diaz

Most department stores offer some sort of layaway plan. Layaway plans differ from credit plans in that no interest charge is added to the sales price, and that delivery of the merchandise is not made until the account is paid in full.

When a layaway transaction occurs, the merchandise is stored away with a tag attached to it. The tag indicates the buyer's name and address, the sales price and the balance due, and any other pertinent information the store wishes to include. The customer usually must make a down payment on a layaway plan. The customer then makes periodic installment payments which the store subtracts from the balance due. Upon the last payment, the merchandise is delivered, and title is transferred to the customer.

Two aspects of the layaway plan are that custody of the merchandise remains with the vendor until the account is paid in full, and that there is no predetermined installment amount or payment date. The store does, however, usually set a date by which the account must be paid in full.

I wrote this program, Layaway, (see Program Listing) for a TRS-80 Model III with at least one disk drive unit, a Disk Operating System, and a parallel printer. As written, the program can handle approximately 300 accounts per disk. To increase this number, label each additional disk from A to Z and make each label number part of the account code. For example, F-145 is located on the disk labelled F. With this approach, you can keep an infinite number of accounts with no need to invest in additional disk drives. The program also has an automatic purge function that places new records into slots of inactive records.

Layplan is the only data file the programs requires. Data records on this file are shown in Table 1.

How the Program Works

Since the program uses only one data file open, use just one buffer when you enter Disk BASIC. In TRSDOS, enter 1 in answer to the HOW MANY FILES prompt. When you run the program, it asks for a system date if you have not entered one via the DOS command DATE. This does not work on the Models I and II. Lines 50–70 display the program menu (Figure 1). File initialization is accomplished in lines 840–900. The report heading routine

Position	Description	Field Name	Length
1	Account code	CODE$	4
2	Customer name	NAM$	30
3	Address line 1	AL$	30
4	Address line 2	BL$	30
5	Telephone number	TELE$	8
6	Identification	SS$	11
7	Driver's license number	LIC$	9
8	Employer name	JOB$	30
9	Description	DESC$	50
10	Debits (charges)	DR$	8
11	Credits (payments/adjustments)	CR$	8
12	Transaction date	PO$	8
13	Expected date of delivery	PU$	8
14	Last payment date	LP$	8

Table 1. *Data records*

starts in line 920. As written, the program outputs five line feeds to set the paper to the appropriate heading titles (top of form). If this is unsuitable to you, modify this line as required or LPRINT either CHR$(12) or CHR$(140) to generate a page advance on the printer.

Upon the menu display, if you type END in response to the prompt ENTER YOUR OPTION (END), program control goes to line 910 where program execution terminates properly.

To create a new account record, select option 1 from the menu (Figure 2). This directs program control to lines 100–210 which contain the routines in which you enter the new account data. The program determines which is the highest available record on the file and assigns its number as the

```
        *** LAY-A-WAY PLAN ***
1. CREATE NEW ACCOUNT
2. LIST OF ACCOUNTS
3. ENTER PAYMENTS
4. ENTER ADJUSTMENTS
5. ACCOUNT STATUS
6. PAST DUE ACCOUNTS
     SYSTEM DATE : 04/01/82
ENTER YOUR OPTION(END)___
```

Figure 1. *Program menu*

```
CREATE NEW LAYAWAY ACCOUNT...

NAME (END) : JOHN DOE
ADDRESS LINE #1 : LAS AMERICAS AVENUE
ADDRESS LINE #2 : PONCE, PUERTO RICO 00731
TELEPHONE (NNN-NNNN) : 555-1212
SOCIAL SECURITY NUMBER (SSS-SS-SSSS) : 123-45-6789
DRIVER'S LICENSE # : 987-65-4321
PLACE OF WORK : ABC COMPUTER CORPORATION
ITEM DESCRIPTION : ONE 25" COLOR TELEVISION

PICK-UP DATE (MM/DD/YY) : 02/24/82
SALE AMOUNT : 750.50
DOWN PAYMENT : 125.00
CORRECT (Y/N) ____
```

Figure 2. *Option 1, create a new account*

customer account code. If the number encountered is greater than the default value of 300, the program halts, asks you to use a new disk, and then returns you to the menu. Upon successful completion of the data entry for each account, the routine in lines 290–340 generates a printed customer data sheet. (See Figure 3.)

```
CUSTOMER DATA SHEET

ACCOUNT CODE : 1
NAME : JOHN DOE
ADDRESS : LAS AMERICAS AVENUE
          PONCE, PUERTO RICO 00731
TELEPHONE : 555-1212
SOCIAL SECURITY : 123-45-6789
LICENSE : 987-65-4321
PLACE OF WORK : ABC COMPUTER CORPORATION
DESCRIPTION : ONE 25" COLOR TELEVISION
PURCHASE DATE : 12/31/81
PICK-UP DATE : 02/24/82
GROSS AMOUNT : 750.50
PAYMENTS : 125.00
BALANCE DUE : 625.00
```

Figure 3. *Customer data sheet*

LIST OF ACCOUNTS . . .
HIT (ENTER) WHEN PRINTER IS READY ____

LIST OF ACCOUNTS PAGE : 1

CODE	NAME	SOCIAL SECURITY	DEBITS	CREDITS	BALANCE
1	JOHN DOE	123-45-6789	750.50	125.00	625.50
2	SMITH, FREDERICK W.	987-65-4321	75.27	10.00	65.27
TOTALS			$825.77	$135.00	$690.77

Figure 4. *Option 2, listing of all accounts*

ENTER PAYMENTS . . .
ENTER CUSTOMER CODE (END) : 1
JOHN DOE
IS THIS CORRECT? (Y/N) Y

CHECK NUMBER (OR CASH) 123
AMOUNT PAID : 10.00
CORRECT? (Y/N) ____

ABOUT TO PRINT CASH RECEIPT SLIP . . .
HIT (ENTER) WHEN PRINTER IS READY . . ____

Figure 5. *Option 3, entering payments*

CASH RECEIPT SLIP

DATE : 12/31/81

RECEIVED FROM : JOHN DOE
LAS AMERICAS AVENUE
PONCE, PUERTO RICO 00731 123-45-6789

THE AMOUNT OF 10.00 TO BE CREDITED TO ACCOUNT NUMBER 1
REFERENCE : 123

RECEIVED BY

THANK YOU.

Figure 6. *Output from option 3*

To obtain a complete listing of all customer accounts on a disk, select option 2 from the menu (Figure 4). Lines 220–280 print this report which shows the account code, customer name, identification code, sale price, payments made to date, and current balance due.

Select option 3 to enter payments (Figure 5). The routine to accept payment data is contained in lines 350–490. A cash receipt slip (Figure 6) is produced upon successful completion of each payment data entry session.

Selecting option 4 allows you to enter adjustments. This routine is contained in lines 500–590. If you want to enter a credit amount as an adjustment, your input must be negative.

An account status query is handled by option 5 (Figure 7). Lines 600–680 contain this routine. A full account status is displayed for requested ac-

```
CUSTOMER ACCOUNT STATUS

NAME : JOHN DOE
ADDRESSS : LAS AMERICAS AVENUE
           PONCE, PUERTO RICO 00731
TELEPHONE : 555-1212                    SOCIAL SECURITY : 123-45-6789
LICENSE : 987-65-43  EMPLOYER : ABC COMPUTER CORPORATION
DESCRIPTION : ONE 25" COLOR TELEVISION
PURCHASE DATE : 12/31/81                PICK-UP DATE : 02/24/82
GROSS AMOUNT : 750.50
PAYMENTS : 135.00
BALANCE DUE : 615.50

DO YOU WISH A HARDCOPY PRINTOUT? (Y/N) ____
```

Figure 7. *Option 5, account status query*

```
PAST DUE ACCOUNTS     PAGE : 1
AS OF : 04/01/82
CODE
NAME                  PICK-UP   LAST PAID   DEBITS  CREDITS  BALANCE
  1
JOHN DOE              02/24/82  12/31/81    750.50   135.00   615.50
  2
SMITH, FREDERICK W.   03/01/82  12/31/81     75.27    10.00    65.27
TOTALS                                     825.77   145.00   680.77
```

Figure 8. *Option 6, list of overdue accounts*

counts. The routine to send output to the printer is contained in lines 690–740.

The last option available from the menu produces a report of overdue accounts (Figure 8). Use this if you enter an estimated delivery date for the merchandise. Accounts which reflect a delivery date later than the actual system date are not shown. This report contains the account code, customer name, expected pickup date, date the last payment was received, debits, credits, and the balance due.

Program Listing. *Layaway*

Encyclopedia Loader

```
10 REM "LAYAWAY- MIGUEL DIAZ
               P.O. BOX 8475
              PONCE, P.R.  00732      ..... 10/81
20 REM " > PROGRAM INITIALIZATION <<<<<<<<<<<<<<<<<<<<<<<<<<
30 CLEAR5000:DEFINTA-Z:DEFDBLS,P,C,D,T:M$="######.##-":M1$="$#####.##-
    ":DIMK(300)
40 D$=LEFT$(TIME$,8):IFD$<>"00/00/00"THEN50ELSECLS:LINEINPUT"ENTER SYS
    TEM DATE (MM/DD/YY) : ";D$:IFLEN(D$)<>8THEN40
50 CLS:PRINTCHR$(23)"*** LAY-A-WAY PLAN ***":PRINT:PRINT"1. CREATE NEW
     ACCOUNT":PRINT"2. LIST OF ACCOUNTS":PRINT"3. ENTER PAYMENTS":PRI
    NT"4. ENTER ADJUSTMENTS"
60 PRINT"5. ACCOUNT STATUS":PRINT"6. UNDELIVERED ACCOUNTS":PRINT
70 PRINT" SYSTEM DATE : ";D$:PRINT:LINEINPUT"ENTER YOUR OPTION (END) "
    ;O$:IFO$="END"THEN910
80 PO=0:LO=0:O=0:VAL(O$):IFO<1ORO>6THEN50ELSEIFX9=0GOSUB860
90 ONOGOTO110,230,360,510,610,760
100 REM " > CREATE NEW ACCOUNT <<<<<<<<<<<<<<<<<<<<<<<<<<<
110 CLS:PRINT"CREATE NEW LAYAWAY ACCOUNT...":PRINTSTRING$(60,131):LINE
    INPUT"NAME (END) : ";N$:IFN$="END"THEN50
120 LINEINPUT"ADDRESS LINE #1 : ";A1$:LINEINPUT"ADDRESS LINE #2 : ";A2
    $:LINEINPUT"TELEPHONE (NNN-NNNN) : ";T$:LINEINPUT"SOCIAL SECURITY
     NUMBER (SSS-SS-SSSS) : ";S$
130 LINEINPUT"DRIVER'S LICENCE # : ";L$:LINEINPUT"PLACE OF WORK : ";W$
    :LINEINPUT"ITEM DESCRIPTION : ";I$
140 PRINT:LINEINPUT"PICK-UP DATE (MM/DD/YY) : ";PI$:IFLEN(PI$)<>8THEN1
    40
150 LINEINPUT"SALE AMOUNT : ";SA$:SA=VAL(SA$):IFSA=<0THEN150
160 LINEINPUT"DOWN PAYMENT : ";PA$:PA=VAL(PA$):IFPA<0THEN160
170 PRINT:LINEINPUT"CORRECT (Y/N) ";Y$:IFY$="N"THEN210
180 IFY$<>"Y"THEN170ELSELETC=LOF(1)+1
190 FORL=1TOLOF(1):IFK(L)=0LETC=L:GOTO 205
200 NEXT L
205 GOSUB 850 : PUT 1,C : K(C)=1 : GOTO 290
210 IFC=300THEN940ELSELETN$="":A1$="":A2$="":T$="":S$="":L$="":W$="":I
    $="":SA=0:PA=0:PI$="":C=0:GOTO110
220 REM " > LIST OF ACCOUNTS <<<<<<<<<<<<<<<<<<<<<<<<<<<<
230 CLS:PRINT"LIST OF ACCOUNTS...":PRINT:LINEINPUT"HIT (ENTER) WHEN PR
    INTER IS READY ... ";Z$:GOSUB280
240 FORJ=1TOLOF(1):GET1,J:IFK(J)=0NEXTJ:GOTO270
250 C=CVI(CODE$):DR=CVD(DR$):CR=CVD(CR$):T1=T1+DR:T2=T2+CR:LO=LO+1:IFL
    O>50GOSUB280
260 LPRINTUSING"####";C;:LPRINT"  ";NAM$;"  ";SS$;" ";:LPRINTUSINGM$;D
    R;CR;DR-CR:NEXTJ
270 LPRINT:LPRINT"TOTALS";TAB(50)USINGM1$;T1;T2;T1-T2:T1=0:T2=0:LO=0:P
    O=0:GOTO50
280 GOSUB930:LPRINT"LIST OF ACCOUNTS              PAGE : ";PO:LPRINT:LP
    RINT"CODE  NAME";TAB(35)"SOCIAL SECURITY  DEBITS  CREDITS  BAL
    ANCE":LPRINT:RETURN
290 CLS:PRINT"ABOUT TO PRINT DATA SHEET...":PRINT:LINEINPUT"HIT (ENTER
    ) WHEN PRINTER IS READY ...";Z$:GOSUB930
300 LPRINT"CUSTOMER DATA SHEET":LPRINT:LPRINTSTRING$(70,"*"):LPRINT:LP
    RINT
310 LPRINT"ACCOUNT CODE : ";USING"###";C:LPRINT:LPRINT"NAME : ";N$:LPR
    INT:LPRINT"ADDRESS : ";A1$:LPRINTTAB(10)A2$:LPRINT
320 LPRINT"TELEPHONE : ";T$:LPRINT:LPRINT"SOCIAL SECURITY : ";S$:LPRIN
    T:LPRINT"LICENCE : ";L$:LPRINT:LPRINT"PLACE OF WORK : ";W$
330 LPRINT:LPRINT"DESCRIPTION : ";I$:LPRINT:LPRINT"PURCHASE DATE : ";D
    $:LPRINT:LPRINT"PICK-UP DATE : ";PU$:LPRINT:LPRINT"GROSS AMOUNT :
     ";USINGM$;SA:LPRINT
340 LPRINT"PAYMENTS : ";USINGM$;PA:LPRINT:LPRINT"BALANCE DUE : ";USING
    M$;SA-PA:LPRINT:LPRINTSTRING$(75,"*"):LPRINT:GOTO210
350 REM " > ENTER PAYMENTES <<<<<<<<<<<<<<<<<<<<<<<<<<<<<
360 CLS:PRINT"ENTER PAYMENTS...":LINEINPUT"ENTER CUSTOMER CODE (END) :
     ";C$:IFC$="END"THEN50
370 C=VAL(C$):IFC>LOF(1)THEN360ELSEGET1,C
380 PRINTTAB(10)NAM$:LINEINPUT"IS THIS CORRECT? (Y/N) ";Y$:IFY$="N"THE
```

Program continued

```
    N360ELSEPRINTSTRING$(60,131)
390 LINEINPUT"CHECK NUMBER (OR CASH) ";R$:IFR$=""THEN390
400 LINEINPUT"AMOUNT PAID : ";PA$:PA=VAL(PA$):IFPA<=0THEN400
410 PRINT:LINEINPUT"CORRECT? (Y/N) ";Y$:IFY$="N"THEN440
420 IFY$<>"Y"THEN410ELSEGOTO460
430 LETPA=PA+CVD(CR$):LSETCR$=MKD$(PA):LSETLP$=D$:PUT1,C
440 PA=0:C=0:PA$="":R$="":C$="":GOTO360
450 REM"PRINT CASH RECEIPT SLIP...
460 CLS:PRINT"ABOUT TO PRINT CASH RECEIPT SLIP...":PRINT:LINEINPUT"HIT
    (ENTER) WHEN PRINTER IS READY ... ";Z$:GOSUB930
470 LPRINT"CASH RECEIPT SLIP":LPRINT:LPRINT:LPRINTTAB(45)"DATE : ";D$:
    LPRINT:LPRINT"RECEIVED FROM : ";NAM$:LPRINTTAB(16)AL$:LPRINTTAB(1
    6)BL$;TAB(40)SS$
480 LPRINT:LPRINT:LPRINT"THE AMOUNT OF ";USINGM$;PA;:LPRINT" TO BE CRE
    DITED TO ACCOUNT NUMBER ";C
490 LPRINT:LPRINT"REFERENCE : ";R$:LPRINTTAB(40)"----------
    ----------":LPRINTTAB(45)"RECEIVED BY ":LPRINT"THANK YOU.":LPR
    INT:LPRINTSTRING$(75,"*"):LPRINT:GOTO430
500 REM " > ENTER ADJUSTMENTS <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
510 CLS:PRINT"ENTER ADJUSTMENTS...":LINEINPUT"ENTER CUSTOMER CODE (END
    ) : ";C$:IFC$="END"THEN50
520 C=VAL(C$):IFC>LOF(1)THEN510ELSEGET1,C
530 PRINTTAB(10)NAM$:LINEINPUT"IS THIS CORRECT? (Y/N) ";Y$:IFY$="N"THE
    N510ELSEPRINTSTRING$(60,131)
540 LINEINPUT"ADJUSTMENT REFERENCE ";R$:IFR$=""THEN540
550 LINEINPUT"AMOUNT : ";PA$:PA=VAL(PA$):IFPA=0THEN550
560 PRINT:LINEINPUT"CORRECT? (Y/N) ";Y$:IFY$="N"THEN590
570 LETPA=PA+CVD(CR$):LSETCR$=MKD$(PA):PUT1,C:GOTO590
580 IFP1>0LETPA=PA+CVD(DR$):LSETDR$=MKD$(PA):PUT1,C
590 PA=0:C=0:PA$="":R$="":C$="":GOTO510
600 REM " > PRINT ACCOUNT STATUS <<<<<<<<<<<<<<<<<<<<<<<<<<
610 CLS:PRINT"ACCOUNT STATUS":LINEINPUT"ENTER CUSTOMER CODE (END) : ";
    C$:IFC$="END"THEN50
620 C=VAL(C$):IFC>LOF(1)THEN610ELSEGET1,C:SA=CVD(DR$):PA=CVD(CR$)
630 CLS:PRINT"CUSTOMER ACCOUNT STATUS":PRINTSTRING$(60,"*")
640 PRINT"NAME : ";NAM$:PRINT"ADDRESS : ";AL$:PRINTTAB(10)BL$
650 PRINT"TELEPHONE : ";TELE$,"SOCIAL SECURITY : ";SS$:PRINT"LICENCE :
     ";LI$;" EMPLOYER : ";JOB$:PRINT"DESCRIPTION : ";DESC$
660 PRINT"PURCHASE DATE : ";PO$,"PICK-UP DATE : ";PU$:PRINT"GROSS AMOU
    NT : ";USINGM$;SA
670 PRINT"PAYMENTS : ";USINGM$;PA:PRINT"BALANCE DUE : ";USINGM$;SA-PA:
    PRINTSTRING$(60,"*")
680 LINEINPUT"DO YOU WISH A HARDCOPY PRINTOUT? (Y/N) ";Y$:IFY$="N"THEN
    610
690 CLS:PRINT"ABOUT TO PRINT ACCOUNT STATUS...":PRINT:LINEINPUT"HIT (E
    NTER) WHEN PRINTER IS READY ...";Z$:GOSUB930
700 LPRINT"CUSTOMER ACCOUNT STATUS":LPRINT:LPRINTSTRING$(70,"*"):LPRIN
    T:LPRINT
710 LPRINT"ACCOUNT CODE : ";USING"###";C:LPRINT:LPRINT"NAME : ";NAM$:L
    PRINT:LPRINT"ADDRESS : ";AL$:LPRINTTAB(10)BL$:LPRINT
720 LPRINT"TELEPHONE : ";TELE$:LPRINT:LPRINT"SOCIAL SECURITY : ";SS$:L
    PRINT:LPRINT"LICENCE : ";LIC$:LPRINT:LPRINT"PLACE OF WORK : ";JOB
    $
730 LPRINT:LPRINT"DESCRIPTION : ";DESC$:LPRINT:LPRINT"PURCHASE DATE :
    ";PO$:LPRINT:LPRINT"PICK-UP DATE : ";PU$:LPRINT:LPRINT"GROSS AMOU
    NT : ";USINGM$;SA:LPRINT
740 LPRINT"PAYMENTS : ";USINGM$;PA:LPRINT:LPRINT"BALANCE DUE : ";USING
    M$;SA-PA:LPRINT:LPRINTSTRING$(75,"*"):LPRINT:GOTO50
750 REM " > UNDELIVERED ACCOUNTS REPORT <<<<<<<<<<<<<<<<<<<<<<
760 CLS:PRINT"LIST UNDELIVERED ACCOUNTS...":PRINT:LINEINPUT"HIT (ENTER
    ) WHEN PRINTER IS READY ... ";Z$:GOSUB830
770 DU$=RIGHT$(D$,2)+LEFT$(D$,2)+MID$(D$,4,2)
780 FORJ=1TOLOF(1):GET1,J:IFK(J)=0NEXTJ:GOTO820
790 PD$=RIGHT$(PU$,2)+LEFT$(PU$,2)+MID$(PU$,4,2):IFPD$>DU$NEXTJ:GOTO82
    0
800 C=CVI(CODE$):DR=CVD(DR$):CR=CVD(CR$):T1=T1+DR:T2=T2+CR:LO=LO+3:IFL
    O>50GOSUB830
810 LPRINTUSING"####";C:LPRINTNAM$;" ";PU$;"  ";LP$;"    ";:LPRINTUSING
    M$;DR;CR;DR-CR:LPRINT:NEXTJ
820 LPRINT:LPRINT"TOTALS";TAB(49)USINGM$;T1;T2;T1-T2:T1=0:T2=0:LO=0:PO
```

```
    =0:GOTO50
830 GOSUB930:LPRINT"UNDELIVERED ACCOUNTS            PAGE : ";PO:LPRIN
    T"AS OF : ";D$:LPRINT:LPRINT"CODE":LPRINT"NAME";TAB(30)"PICK-UP
    LAST PAID    DEBITS    CREDITS    BALANCE":LPRINT:RETURN
840 REM " > LAYPLAN FILE INTIALIZATION <<<<<<<<<<<<<<<<<<<<<<
850 LSETCODE$=MKI$(C):LSETNAM$=N$:LSETAL$=A1$:LSETBL$=A2$:LSETTELE$=T$
    :LSETSS$=S$:LSETLIC$=L$:LSETJOB$=W$:LSETDESC$=I$:LSETDR$=MKD$(SA)
    :LSETCR$=MKD$(PA):LSETPO$=D$:LSETPU$=PI$:LSETLP$=D$:RETURN
860 CLS:PRINT"READING DATA FILE...":PRINT
870 OPEN"R",1,"LAYPLAN":FIELD1,4 AS CODE$,30 AS NAM$,30 AS AL$,30 AS B
    L$,8 AS TELE$,11 AS SS$,9 AS LIC$,30 AS JOB$,50 AS DESC$,8 AS DR$
    ,8 AS CR$,8 AS PO$,8 AS PU$,8 AS LP$:X9=9
880 FORL=1TOLOF(1):GET1,L:PRINTL;:
890 DR=CVD(DR$):CR=CVD(CR$):IFDR-CR<>OLETK(L)=1ELSEK(L)=0
900 NEXTL:RETURN
910 CLS:CLOSE:CLEAR50:END
920 REM " > TOP OF FORM (PAPER ADVANCE) ROUTINE <<<<<<<<<<<<
930 FORL=1TO5:LPRINT:NEXTL:LO=0:PO=PO+1:RETURN
940 CLS:PRINT"THERE ARE 300 DATA RECORDS ON FILE...":PRINT:PRINT"THIS
    IS THE MAXIMUM ALLOWABLE PER DISC.":PRINT:PRINT"INSERT ANOTHER DI
    SC"
950 PRINT:PRINT"REMEMBER TO LABEL IT!":CLOSE:X9=0:PRINT:LINEINPUT"HIT
    (ENTER) WHEN READY...";Z$:GOTO50
```

Your Fair Share

by Nate Salsbury

A current popular savings method is investing in money market mutual funds to obtain higher interest rates. Most of these funds require an initial deposit of $1000 or more.

When the statement from your fund arrives showing that your account has earned $143.82 in interest, have you ever wondered how to distribute it equitably among the several purposes for which you have made deposits at various times during the period? This program (see Program Listing) allows you to apportion your earnings to your mental subaccounts. As a bonus, you are able to evaluate the effective annual rate of interest as if your funds had been deposited in a savings account with a steady rate of interest, compounded daily over the period being reviewed. You can then compare this rate with that offered by other investment opportunities.

This program covers the various aspects of the problem in their simplest form. You can modify or expand it to meet your personal requirements or to add some bells and whistles to the screen display.

Most current types of investment offer daily compounding of interest. To keep track of this, you need a method of establishing the number of days between any two calendar dates. The subroutine in lines 1000–1130 determines the number of days from the start of a year to any date in that year. The date is expressed numerically. March 27, for example, is represented by 3,27. When you know the day of the year for each of the two dates, the number of days between them is obtained by simple subtraction.

Lines 1000–1030 ensure that M, the number which has been INPUT for the month, is a number from 1 to 12, while lines 1040–1060 ensure that the day of the month, variable D, is a number from 1 to 31. There are 12 items, one for each month, in the DATA list at line 1070. Each entry represents the number of days in the year up to the start of the month you are dealing with. For example, the third entry shows that there are 59 days in the year up to the beginning of March. Forget about leap years. For the dollar amounts you are working with, the difference is insignificant.

Lines 1080–1120 read the number of days up to the start of month, M, then add D days to obtain the number of days from the start of the year to the specified date. This result is stored in variable DN. Next, RESTORE allows you to use the routine over.

Calculating Interest

Suppose that you open an account on the first of January with a $1000 deposit. On July 1, you make a deposit of $1000. To simplify the analysis, assume that this is a savings account which has a constant interest rate throughout the year. You can calculate the total interest earned from these transactions by these two methods.

- The first method is to calculate the interest earned in the first six months on the initial $1000 deposit. On July 1, add that interest plus the second deposit of $1000 to the original $1000 to obtain a balance of $2000 plus the interest earned to date. This is the figure you would find in a savings passbook after the July 1 deposit of $1000. Now, calculate the interest which would be earned on this total to the end of the year. Add this interest to the amount earned up to June 30 and you have the total interest earned for the year.
- Another method is to calculate the interest earned by the initial $1000 deposit for a full year, then calculate the interest earned on the second $1000 deposit for six months. Add the two results. The result is identical to the result obtained by the first method.

You can analyze the money on deposit at the start of the period and each deposit and withdrawal to determine the interest contribution from each one. At the end of a period, you can combine the results arithmetically. I found this approach easier to program.

Suppose that you deposit $200 at the beginning of the period. Then, in the middle of the period, you add $500 intended for another purpose. How do you distribute the interest proceeds between the two causes at the end of the period? For discussion, assume that the period in question is a 30-day month, and these are the only transactions.

The $200 deposit earned interest for the full 30 days, while the $500 deposit earned interest for only 15 days. If you multiply the number of dollars on deposit by the number of days during which they earned interest, you obtain a measure of the weight each deposit had in the final interest payment. In this case, the $200 for 30 days contributed 6000 dollar days ($200 × 30 days), while the $500 deposit contributed 7500 dollar days ($500 × 15 days). The two deposits together, which produced the total interest payment, contributed 13,500 dollar days. Therefore, the account for which the $200 deposit was made receives 6000/13,500 or 44.4 percent of the interest received, while the account for which you deposited $500 receives credit for 7500/13,500 or 55.6 percent of the interest earned.

The concept of multiplying dollars by the number of days invested is a straightforward means of comparing the effects of various transactions made at different times. This same concept applied as a negative result applies to funds withdrawn during the period. The main program outlines the steps I took to implement these concepts. It involves no neat formatting of

the screen displays, no special printouts, and very little error checking of keyboard inputs because you probably want to customize these features for your own situation.

Lines 20–70 establish the day of the year for the starting and ending dates of the period under review. Line 1110 handles periods which include December 31. Lines 80–190 accumulate the opening balance of each account plus all the transactions during the period. This information is stored in array CH( ). The day of the year associated with each entry is stored in array N3( ).

At line 200, the transactions for each account and their accompanying days of the year are stored in master arrays DT( ) and CD( ) by subroutine 2000. Similar entries for the opening balance of each account are made at line 100. Next, lines 210–270 calculate the dollar days for each transaction in the subaccount.

Lines 230–240 accumulate the total dollar days for all transactions in a particular subaccount in variable W. Line 250 calculates the net dollar changes of all transactions in the subaccount. Finally, you calculate the dollar days for the opening balance of each subaccount and add them to the transaction total in line 270. The array variable WT(AN) now contains the total weighting factor of dollar days for the subaccount under consideration.

Line 280 stores the final balance of the account, that is, the balance just before interest is added, and line 290 stores the opening balance. I used this feature in my personal program as part of a report. The array OB( ) is not used in subsequent calculations in this listing.

The intermediate arrays CH( ) and N3( ) are set to zero to prepare for entries associated with the next subaccount in lines 310–330. In line 340, you determine whether there are more subaccounts and, if there are, the program loops back to line 80. Otherwise, the program has received all of the necessary data and continues to line 360 where the final income distribution is made.

All of the dollar days for each account are added together in lines 360–380 to obtain the total of all the weighting factors. After you enter the total interest earned in the period in line 390, the program distributes this income to each account according to your weighting factor analysis and stores the distributed amounts in array ID( ).

Getting Your Fair Share

At this point, the key information developed by the program is available in three arrays where it is easily accessible for reports or calculations. Lines 430–460 outline what data is stored and where it is located. Lines 470–490 print on the screen the final balance in each subaccount after distribution of the interest earned by the total account. Note that the amount shown here as the closing balance is the figure you enter in the next period as the opening balance for each account.

Could I Do Better?

If your investment is in a money market fund, you have already noted that the equivalent annual interest rates fluctuate. Actual current daily rates react rapidly to changes in the prime interest rate. I wanted to determine the effective annual rate which I actually received over any period under review. This is calculated in lines 495–730 to within .1 percent. The process takes as much as 45 seconds if you are enjoying a rate near 20 percent and have a lot of transactions.

In lines 510–540, the program determines the current total value of your account. The program assumes a fixed interest rate, variable I in line 560, then calculates the final value of your account as if all the transactions you made had occurred at that fixed rate. The final result, variable S in line 600, is subtracted from TV, the actual total value. If the result is positive, the assumed interest rate was too low. The rate is then increased, and the process repeats. This is done in a FOR-NEXT loop in lines 560–640 over the range of four percent to 25 percent in increments of one percent.

When the difference becomes negative, the present value of I is one percent too high. In line 660, change the range of the loop to enclose the crossover point and change the step size to .1 percent. Then repeat the whole comparison process. Line 650 is a switch which limits this routine to two iterations. The rest of the main program, lines 690–750, prints out the result of this test.

Using this program, you can distribute your gross interest properly to all your subaccounts and, in addition, track the effective annual interest rate, with daily compounding, which you are receiving.

Program Listing. *Your Fair Share*

Encyclopedia Loader™

```
1 REM    "YOUR FAIR SHARE"
2 REM    by NATE SALSBURY
3 REM    608 Madam Moore's Lane
4 REM    New Bern, NC     28560
5 ON ERROR GOTO 2100
10 CLS
20 INPUT "ENTER START OF PERIOD (MONTH #, DAY)"; M1,D1
25 M = M1: D = D1
30 GOSUB 1000
32 IF FM = 1 THEN M1 = M: FM = 0
34 IF FD = 1 THEN D1 = D: FD = 0
36 SD$ = STR$(M1) + " /" + STR$(D1)
40 N1 = DN: REM N1= DAY OF YEAR WHEN PERIOD STARTS
50 INPUT "ENTER END OF PERIOD (MONTH #, DAY)"; M2,D2
55 M = M2: D = D2
60 GOSUB 1000
62 IF FM = 1 THEN M2 = M: FM = 0
64 IF FD = 1 THEN D2 = D: FD = 0
66 ED$ = STR$(M2) + " /" + STR$(D2)
70 N2 = DN: REM N2 = DAY OF YEAR WHEN PERIOD ENDS
72 CLS: PRINT "HOW MANY ACTIVE ACCOUNTS/CATEGORIES WERE THERE IN THE P
   ERIOD": PRINT "FROM "  SD$ " TO " ED$;: INPUT A1
74 INPUT "ENTER THE TOTAL NUMBER OF TRANSACTIONS (DEPOSITS AND/OR WITH
   DRAWALS) WHICH YOU MADE IN THIS PERIOD   ";A2
76 A3 = 2*A1 + A2: DIM DT(A3), CD(A3), N3(A3), CH(A3)
78 CD(0) = N2
79 REM NEXT 'INPUT' ASSUMES THAT EACH SUBSECTION OF THE FUND IS CODED
   WITH AN ACCOUNT # (1,2,3 ETC.)
80 CLS: INPUT "ENTER IDENTIFYING ACCOUNT NUMBER FOR THIS PART OF THE F
   UND"; AN
89 REM THE NEXT 'INPUT' REQUESTS THE BALANCE OF THIS SUBACCOUNT AT THE
        END OF THE PREVIOUS REVIEW. IF THERE IS NO PREVIOUS BALANCE,
    ENTER  0
90 PRINT "ENTER BALANCE FOR ACCOUNT #"; AN; "AT THE START OF THIS PERI
   OD"
100 INPUT OB: K = K + 1: DT(K) = OB: CD(K) = N1
110 PRINT "DID YOU ADD OR DEDUCT ANY AMOUNTS FROM ACCOUNT #" AN: PRINT
     "IN THIS    PERIOD (Y / N)"; : INPUT Y$
120 IF Y$ = "N" GOTO 200
125 IF Y$ <> "Y" THEN 110
130 PRINT "ENTER (MONTH #, DAY) WHEN A CHANGE WAS MADE TO ACCOUNT #";
   AN : INPUT M,D
140 GOSUB 1000: N3 = DN: IF N1<=N3 AND N3<=N2 THEN 150
145 PRINT STR$(M) + " /" + STR$(D) " IS NOT IN THE PERIOD FROM" SD$ "
   TO " ED$: PRINT: PRINT"PLEASE REENTER IT": PRINT: GOTO 130
150 INPUT "ENTER THE DOLLAR CHANGE MADE ON THIS DATE.  (SHOW WITHDRAWA
   LS    WITH A MINUS SIGN E,G, -500.75)"; CH
160 I = I + 1: REM I = ITEM # OF A CHANGE IN THIS ACCOUNT IN THIS PERI
   OD
170 N3(I) = N3: CH(I) = CH
180 PRINT: PRINT "ARE THERE ANY OTHER ADDITIONS AND/OR WITHDRAWALS TO
   ACCOUNT #"; AN; "IN THIS PERIOD (Y / N)": INPUT Y$
190 IF Y$ = "Y" THEN PRINT: GOTO 130
195 IF Y$ <> "N" THEN 180
199 REM AT THIS POINT, ALL CHANGES TO THE ACCOUNT HAVE BEEN STORED IN
   ARRAYS
200 GOSUB 2000: REM THIS STORES THE ACCOUNT DATE IN THE MASTER ARRAY
209 REM LINES 210 - 290 CALCULATE 'DOLLARS * DAYS' FOR EACH TRANSACTIO
   N AND KEEP A RUNNING BALANCE FOR THE ACCOUNT, NEGLECTING INTEREST
    ACCUMULATION
210 W=0: X=0
220 FOR A = 1 TO I
230 Y = CH(A) * (N2 - N3(A)): REM CALCULATE 'DOLLARS * DAYS' FOR EACH
   CHANGE IN THE ACCT
240 W = W + Y: REM CUMULATIVE 'DOLLARS * DAYS' FOR ALL CHANGES
250 X = X + CH(A): REM TOTAL OF ALL DOLLAR CHANGES TO ACCOUNT FROM ADD
   ITIONS AND/OR WITHDRAWALS
```

```
260 NEXT A
269 REM CALCULATE 'DOLLARS * DAYS' FOR OPENING BALANCE AND ACCUMLATED
     TOTAL FOR ALL CHANGES
270 WT(AN) = OB * (N2 - N1) + W
280 B(AN) = OB + X: REM B(AN) IS THE FINAL BALANCE IN THE ACCOUNT, IGN
     ORING INTEREST
290 OB(AN) = OB
300 I = 0
310 FOR A = 1 TO A3
320 CH(A) = 0: N3(A) = 0
330 NEXT A
340 INPUT "ARE THERE ANY MORE ACCOUNTS (Y / N)";Y$
350 IF Y$ = "Y" THEN 80
355 IF Y$ = "N" THEN CLS ELSE 340
359 REM FINAL PROCESSING - DISTRIBUTION OF INCOME
360 FOR A = 1 TO 10
370 WF = WF + WT(A)
380 NEXT A
390 PRINT: PRINT: PRINT: INPUT "ENTER TOTAL INTEREST RECEIVED IN THIS
     PERIOD"; TI
400 FOR A = 1 TO 10
410 ID(A) = (WT(A) / WF) * TI
420 NEXT A
430 REM AT THIS POINT, ALL PERTINENT ACCOUNT INFORMATION IS STORED IN
     ARRAYS
440 REM OB() = BALANCE IN THE ACCOUNT AT START OF PERIOD
450 REM B() = BALANCE IN THE ACCOUNT AT END OF PERIOD EXCEPT FOR INTER
     EST
460 REM ID() = DISTRIBUTION OF INTEREST TO EACH ACCOUNT
470 CLS: PRINT @ 64, "ACCOUNT NUMBER", , "CLOSING BALANCE"
480 FOR A = 1 TO 10
485 PRINT TAB(4) USING "##"; A; : PRINT ,, USING "$ ##,###.##"; B(A) +
     ID(A)
490 NEXT A
495 PRINT: PRINT TAB(16) "CALCULATING INTEREST";
500 REM FROM THIS POINT, THE PROGRAM CALCULATES THE 'EQUIVALENT ANNUAL
     INTEREST RATE' BASED ON THE INTEREST JUST RECORDED, THE VARIOUS
     TRANSACTIONS IN THE ACCOUNT AND THE LENGTHS OF TIME THEY WERE INV
     ESTED AND, THEREFORE, EARNED INTEREST.
510 FOR A = 1 TO 10
520 V = B(A) + ID(A): REM THIS IS THE FINAL VALUE IN EACH ACCT AFTER I
     NTEREST DISTRIBUTION
530 TV = TV + V: REM THIS IS THE TOTAL VALUE OF THE FUND
540 NEXT A
550 AL = .04: AH = .25: C = .01
560 FOR I = AL TO AH STEP C
570 FOR J = 1 TO K: REM K = TOTAL NUMBER OF ALL TRANSACTIONS
580 D = CD(0) - CD(J)
590 A = DT(J) * (1 + I/365) [D
600 S = S + A
610 NEXT J
620 IF TV - S > 0 ELSE GOTO 650
630 S = 0
640 NEXT I
650 Q = Q + 1: IF Q = 2 GOTO 690
660 AL = I - .01: AH = I: C = .001
670 S = 0
680 GOTO 560
690 PRINT: PRINT "IF THE INTEREST RATE HAD BEEN CONSTANT OVER THIS PER
     IOD, THE     EFFECTIVE ANNUAL RATE  WOULD HAVE BEEN"
710 PRINT TAB(5) "BETWEEN      "; (I - .001) * 100; "AND" ; I * 100; "
     %  COMPOUNDED DAILY";
720 REM IF THE RANGE SHOWN AT LINE 710 IS  '25 TO 25.1 % '  THEN THE A
     CTUAL RATE IS MORE THAN 25 %
730 REM IF THE RANGE SHOWN AT LINE 710 IS  '2.9 TO 3% ' THEN THE ACTUA
     L RATE IS LESS THAN 4%
740 GOTO 740
750 END
1000 REM CALCULATE NUMBER OF DAYS FROM START OF YEAR TO DATES INPUT FR
     OM KEYBOARD ( 'M' FOR MONTH NUMBER, 'D' FOR DAY OF THE MONTH)
```

Program continued

```
1009 REM CHECK HERE FOR VALID MONTH NUMBER
1010 IF M > 0 AND M < 13 GOTO 1040
1020 PRINT M; "IS NOT A MONTH NUMBER. PLEASE CORRECT": FM = 1
1030 INPUT M: GOTO 1010
1035 REM CHECK HERE FOR VALID DAY OF THE MONTH
1040 IF D > 0 AND D < 32 GOTO 1080
1050 PRINT D; "IS NOT A DAY IN ANY MONTH.  PLEASE CORRECT": FD = 1
1060 INPUT D: GOTO 1040
1065 REM DATA ITEMS ARE THE NUMBER OF DAYS FROM START OF THE YEAR TO T
     HE END OF THE PRECEDING MONTH, FOR EACH MONTH OF THE YEAR
1070 DATA 0,31,59,90,120,151,181,212,243,273,304,334
1080 FOR A = 1 TO M
1090 READ DN: NEXT A
1100 DN = DN + D: REM ADD # OF DAYS IN CURRENT MONTH
1110 IF DN < N1 THEN DN = DN + 365: REM CORRECTION IF DEC 31 IS IN THE
      PERIOD
1120 RESTORE
1130 RETURN
2000 REM TRANSFER ALL INDIVIDUAL CHANGES TO MASTER ARRAY
2010 FOR A = 1 TO I: REM I = # ITEMS ENTERED FOR A SINGLE ACCT
2020 K = K + 1: REM K = TOTAL NUMBER OF ALL CHANGES
2030 DT(K) = CH(A): REM STORE $ CHANGES
2040 CD(K) = N3(A): REM STORE DAY OF THE YEAR NUMBER
2050 NEXT A: RETURN
2100 IF ERR/2 + 1 = 9 THEN 2120
2110 ON ERROR GOTO 0: RESUME
2120 CLS: PRINT @ 512, "TOO MANY ENTRIES.  REVIEW DATA AND RERUN."
2130 ON ERROR GOTO 0: RESUME
```

EDUCATION

Do-It-Yourself Maze Package
Getting Your Bearings

Do-It-Yourself Maze Package

by Anne Weiss

There is an abundance of maze and adventure games available for computer users. Many of them are quite good, highly involved, take days or weeks to solve, and are very expensive. If you love and can afford such programs, this article is probably not for you. But, if you are involved with teaching a much larger number of students than the number of computers available, this might be of interest.

We at St. Peter's High School were in just such a situation. There are two Model I 16K Level II Radio Shack computers to accommodate between 15 and 20 students at a time. Part of the problem is alleviated somewhat by scheduling each student into at least one 40-minute lab period per week. That way, we can guarantee weekly computer time with two students to a machine. That solution seemed short-lived when 32 seniors signed up for the computing course. Since I couldn't tell them to wait until next year, and I didn't want to turn them away, I had to do some improvising. The first step was to divide the course into two one-semester courses.

While the split solved the numbers problem, it added another problem—that of concentrating 10 months worth of four classes and a lab into five months. I didn't want the course to become heavily structured with very little student involvement. The trick then was to consolidate without taking the life out of the course. I developed the following package as one means towards a no-frills, meaningful course. The activity is good for students who enjoy programming and also those who just want to know how to use the computer. To use this package, the students should know the start-up sequence, how to use PRINT statements, and how to use the cassette recorder.

A word of caution: The finished product will not set any software publisher's heart to pounding. The package is meant as a fun exercise to open doors into methods of computing. The students get to see a variety of BASIC expressions at work (REM, LET, FOR-NEXT, READ, DATA, string and numeric variables, numeric arrays, ON-GOSUB, RETURN, PRINT, INPUT, LEFT$, IF-THEN, GOTO, END, RUN) along with methods of error checking. Less proficient students know that if they just follow directions, the program will work. The coding sheets make debugging a bit

easier. Even typing teaches the student how to enter information into the computer.

A more creative student can use this exercise as a starting point to design larger and more involved mazes and add treasures and obstacles to the program. My students like to add graphics and flashing screen displays.

Although this activity is considered a package, students do not get all the components at once. Each sheet is given out or described at the appropriate time. I have written the lessons as if they were on individual sheets for each student, but some of the information can be given verbally or on the blackboard. I have also included time, materials, and outcomes for each lesson. I have included the listing for a maze created by Steve Armistead, class of 1981.

Lesson 1: Introduction

Time: One to three sessions, depending on the class size and the number of computers.

Materials: TRS-80 Model I 16K Level II
Copy of Monster's Gold program
Paper and pencil

Outcome: Students become familiar with computer adventures.
Use of the computer is reinforced.
Students work together as a group and use logic to solve a problem.

Load the Monster's Gold program into the computer so it is ready to run. It is a maze game which contains 25 squares. The object is to travel in and out of the squares by moving in one of the four compass directions. Sometimes your path is blocked, and you must change direction. You can also go through a magic door and end up far away from where you were. You might be lucky and find the treasure. Then again, you might fall into a trap!

See if you can solve the puzzle. Try to make a map based on your moves. Remember, there's magic at work here. A single move can bring you into the next square, or maybe 10 squares away. Have fun!

Lesson 2: Following a Maze Map

Time: One session

Materials: Computer
Tape of Monster's Gold Program
Map for Monster's Gold (Figure 1)

Outcome: Learn to read and follow a maze map

Look at the grid in Figure 1. It represents the paths through the Monster's Gold maze. There are four ways to move from each square. The arrows show where you wind up after each move. Blocked exits are marked with a heavy line.

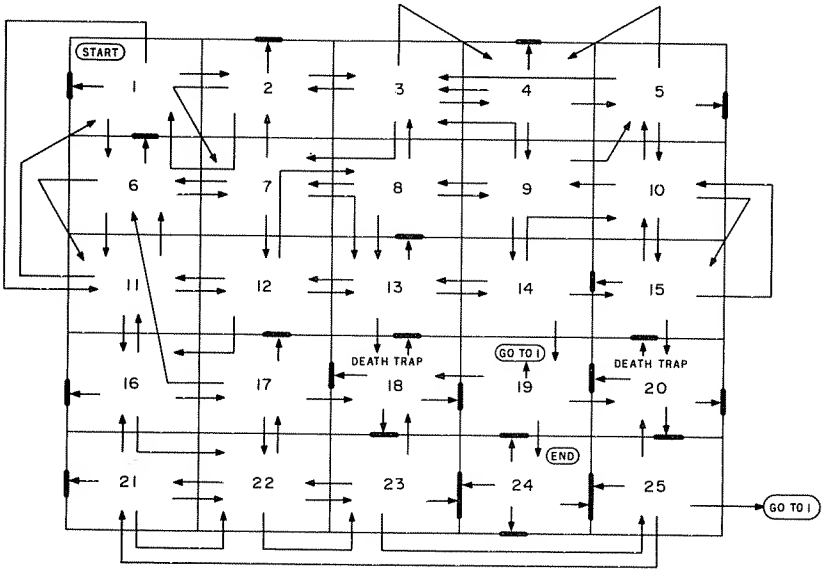


Figure 1. *Monster's Gold map*

Notice that you can move east from square 14 to get into square 15 but you cannot move back from 15 into 14. If you are in square 21, you can get to square 22 by moving either east or south. A move north from square 12 brings you into square 8.

How many times did you fall into the death traps when you were trying to solve the maze? Play the game again, using the map to guide you. Is there more than one solution path? See if you can label each square with a description such as high cave or damp cave.

Lesson 3: Designing a Maze

Time: One session
Materials: Grid sheet (Figure 2) and pencil

It is now time to design your own maze on a grid sheet. Choose the number of the square at which you want your adventure to begin and write *start* in that square. Use arrows to lay out a map of your maze. Remember, each square has four possible exits, N,E,S,W. Use a heavy line to mark any blocked exits. Include death traps or obstacles if you so desire. Label the other squares with a one- or two-word description that goes along with your topic. For example, *Haunted House* could have dungeons, grave yards, and secret halls. *Magic Forest* could have princesses, castles, and streams. *Spooky School* could have labs, lockers, and cafeterias. Let your imagination run wild! When this map is complete, you are ready to start describing the squares in greater detail.

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

Figure 2. *Grid sheet*

Lesson 4: Coding the Map

Time: Two to three sessions
Materials: Monster's Gold Map (Figure 1)
Previously labeled grid sheet (Figure 2)
Coding sheet (Figure 3) and pencil
Outcome: Moves are coded for entrance into the program as DATA.

Refer to the Monster's Gold map in Figure 1. If you were coding the information, it would look like the following example.

Square #	move N to #	move E to #	move S to #	move W to #
1	11	2	6	26
2	26	3	1	7
18	26	26	26	26

The blocked exits are denoted by 26. Notice that square 18 has only 26s, since there is no exit from a death trap. The same would be true for a winning square.

You must now do the same with your map. Note the results of the four directions in each square. Use the coding form in Figure 3 and put the square number at which you wind up after the move. Use 26 to indicate a blocked exit. Include a five- or six-word description of each square.

Lesson 5: Writing the Program

Time: Two to three sessions
Materials: Monster's Gold map (Figure 1)
Previously completed grid and coding form (Figures 2 and 3)
Program form (Figure 4) and pencil

program line #	square #	if move N wind up in #	if move E wind up in #	if move S wind up in #	if move W wind up in #	brief description of square
1000	1					
1010	2					
1020	3					
1030	4					
1040	5					
1050	6					
1060	7					
1070	8					
1080	9					
1090	10					
1100	11					
1110	12					
1120	13					
1130	14					
1140	15					
1150	16					
1160	17					
1170	18					
1180	19					
1190	20					
1200	21					
1210	22					
1220	23					
1230	24					
1240	25					
1250	26					blocked exit

Figure 3. *Maze coding chart*

Fill in your name and the name of your maze in line 10 of the program form. Use PRINT statements in lines 30–130 to give directions.

You are now ready to program your map into the computer. First locate the square marked *start* on your map. Find a square that leads into your start square—that is the initial value of R in line 220. The value of D is the direction needed to move out of R, where N = 1, E = 2, S = 3, W = 4. For an example, look at the Monster's Gold map. The starting square is square 1. In order to get to square 1, you could move west (D = 4) from square 11 (R = 11). Other choices are R = 19 and D = 1, or R = 25 and D = 2. Find an

appropriate choice for R and D in your maze and enter those values in line 220.

```
10  REM ....................................................
20  REM   DESCRIPTION AND DIRECTIONS
30  ....................................................
40  ....................................................
50  ....................................................
60  ....................................................
70  ....................................................
80  ....................................................
90  ....................................................
100 ....................................................
110 ....................................................
120 ....................................................
130 ....................................................
140 '
150 REM INITIALIZE AND READ DATA
160 LET N=25: DIM S(N,4)
170 FOR R=1 TO N
180   FOR D=1 TO 4
190     READ S(R,D)
200   NEXT D
210 NEXT R
220 LET R = .... : LET D = ....
230 '
240 REM MAIN LOOP OF PROGRAM
250 ON S(R,D) GOSUB 1000,1010,1020,1030,1040,1050,1060,1070,1080,1090,1100,1110,
1120,1130,1140,1150,1160,1170,1180,1190,1200,1210,1220,1230,1240,1250
260 IF R=0 THEN 1260
500 PRINT: PRINT
510 INPUT "WHICH WAY WOULD YOU LIKE TO MOVE (N,E,S,W) ";Q$
520 PRINT: Q$=LEFT$(Q$,1)
530 IF Q$="N" THEN D=1: GOTO 250
540 IF Q$="E" THEN D=2: GOTO 250
550 IF Q$="S" THEN D=3: GOTO 250
560 IF Q$="W" THEN D=4: GOTO 250
570 PRINT "NO! NO!  PAY ATTENTION NEXT TIME!": GOTO 510
1000 REM SQUARE 1
1002 ....................................................
     ....................................................
     ....................................................
1005 DATA .... , .... , .... , ....
1009 LET R = .... : RETURN
1010 REM SQUARE 2
1012 ....................................................
     ....................................................
     ....................................................
1015 DATA .... , .... , .... , ....
1019 LET R = .... : RETURN
1020 REM SQUARE 3
1022 ....................................................
     ....................................................
     ....................................................
1025 DATA .... , .... , .... , ....
1029 LET R = .... : RETURN
1030 REM SQUARE 4
1032 ....................................................
     ....................................................
     ....................................................
1035 DATA .... , .... , .... , ....
1039 LET R = .... : RETURN
1040 REM SQUARE 5
1042 ....................................................
     ....................................................
     ....................................................
1045 DATA .... , .... , .... , ....
1049 LET R = .... : RETURN
1050 REM SQUARE 6
```

```
1052 ....................................................
     ....................................................
     ....................................................
1055 DATA .... , .... , .... , ....
1059 LET R = .... : RETURN
1060 REM SQUARE 7
1062 ....................................................
     ....................................................
     ....................................................
1065 DATA .... , .... , .... , ....
1069 LET R = .... : RETURN
1070 REM SQUARE 8
1072 ....................................................
     ....................................................
     ....................................................
1075 DATA .... , .... , .... , ....
1079 LET R = .... : RETURN
1080 REM SQUARE 9
1082 ....................................................
     ....................................................
     ....................................................
1085 DATA .... , .... , .... , ....
1089 LET R = .... : RETURN
1090 REM SQUARE 10
1092 ....................................................
     ....................................................
     ....................................................
1095 DATA .... , .... , .... , ....
1099 LET R = .... : RETURN
1100 REM SQUARE 11
1102 ....................................................
     ....................................................
     ....................................................
1105 DATA .... , .... , .... , ....
1109 LET R = .... : RETURN
1110 REM SQUARE 12
1112 ....................................................
     ....................................................
     ....................................................
1115 DATA .... , .... , .... , ....
1119 LET R = .... : RETURN
1120 REM SQUARE 13
1122 ....................................................
     ....................................................
     ....................................................
1125 DATA .... , .... , .... , ....
1129 LET R = .... : RETURN
1130 REM SQUARE 14
1132 ....................................................
     ....................................................
     ....................................................
1135 DATA .... , .... , .... , ....
1139 LET R = .... : RETURN
1140 REM SQUARE 15
1142 ....................................................
     ....................................................
     ....................................................
1145 DATA .... , .... , .... , ....
1149 LET R = .... : RETURN
1150 REM SQUARE 16
1152 ....................................................
     ....................................................
     ....................................................
1155 DATA .... , .... , .... , ....
1159 LET R = .... : RETURN
1160 REM SQUARE 17
1162 ....................................................
     ....................................................
     ....................................................
1165 DATA .... , .... , .... , ....
```

Program continued

```
1169 LET R = .... : RETURN
1170 REM SQUARE 18
1172 ....................................................
     ....................................................
     ....................................................
1175 DATA .... , .... , .... , ....
1179 LET R = .... : RETURN
1180 REM SQUARE 19
1182 ....................................................
     ....................................................
     ....................................................
1185 DATA .... , .... , .... , ....
1189 LET R = .... : RETURN
1190 REM SQUARE 20
1192 ....................................................
     ....................................................
     ....................................................
1195 DATA .... , .... , .... , ....
1199 LET R = .... : RETURN
1200 REM SQUARE 21
1202 ....................................................
     ....................................................
     ....................................................
1205 DATA .... , .... , .... , ....
1209 LET R = .... : RETURN
1210 REM SQUARE 22
1212 ....................................................
     ....................................................
     ....................................................
1215 DATA .... , .... , .... , ....
1219 LET R = .... : RETURN
1220 REM SQUARE 23
1222 ....................................................
     ....................................................
     ....................................................
1225 DATA .... , .... , .... , ....
1229 LET R = .... : RETURN
1230 REM SQUARE 24
1232 ....................................................
     ....................................................
     ....................................................
1235 DATA .... , .... , .... , ....
1239 LET R = .... : RETURN
1240 REM SQUARE 25
1242 ....................................................
     ....................................................
     ....................................................
1245 DATA .... , .... , .... , ....
1249 LET R = .... : RETURN
1250 REM BLOCK
1252 PRINT "THAT WAY IS BLOCKED!  TRY ANOTHER DIRECTION."
1259 RETURN
1260 REM GAME IS OVER
1270 PRINT: PRINT: PRINT
1280 FOR I=1 TO 1000: NEXT I
1290 INPUT "WOULD YOU LIKE TO PLAY AGAIN (Y OR N) ";Q$
1300 IF LEFT$(Q$,1)="Y" THEN RUN
1310 END
```

Figure 4. *Program form*

There are 25 small segments of the program that you must now fill in on the program form—one for each square. You must fully expand the descriptions of your square to include everything that you want on the screen. In addition to telling about the particular square, you can give

hints or warnings about nearby squares, especially if a death trap or winning square is close. Whatever you want on the screen must be enclosed in quotation marks. Use more lines if necessary, but watch your numbering.

After the PRINT statement is filled in for a particular square, write in the four destination codes in the DATA statements. Notice that there are commas between the numbers, but not at the end of the statement. The destination codes come from the coding sheet you filled out in Figure 3.

You must tell the computer which square you are in using the LET R = ... statement. If the square is a death trap or a winning square, fill in the blank with 0. For all other cases, use the number of the square.

You must fill in the description, DATA, and square number for each of the 25 squares. If you need an example, look at the code for Monster's Gold on the sheet for lesson 4. The following lines would appear in that program:

```
1000 REM SQUARE 1
1002 PRINT "YOU ARE NOW IN A LARGE DARK CAVE. SUDDENLY YOU HEAR"
1003 PRINT "WINGS FLAPPING. OH NO! IT'S A HARPY! GET OUT QUICK"
1005 DATA 11,2,6,26
1009 LET R = 1: RETURN
1170 REM SQUARE 18 (DEATH TRAP)
1172 PRINT "TOO BAD! YOU'VE ALL JUST FALLEN INTO A HUGE FISSURE
     AND DIED"
1175 DATA 26,26,26,26
1179 LET R = 0: RETURN
```

Lesson 6: Entering and Debugging the Program

Time: Three to five sessions
Materials: Computer with cassette tape and recorder
Program form (Figure 4) and pencil

Once you have filled out and checked the program form in Figure 4, you are ready to type the program into the computer. Follow the usual procedures for start-up and enter the program. Watch out for typing errors as you go along. If you do not finish by five minutes before the end of the period, CSAVE your program on your tape. Be sure to write down the limits of the program counter and to check by using CLOAD? The next time you work on the computer, CLOAD the latest version of your program and start typing from there. Everytime you CSAVE, advance the counter five digits and use a new strip of tape. Continue this procedure until you have entered the whole program.

Run the program and use your map and code (Figures 2 and 3) to find and correct any errors. Make corrections on the program form (Figure 4) as well. CSAVE the final version on a new tape.

Lesson 7: Follow-up Lesson

Time: One to three sessions, as time permits

Materials: Computer
Copies of completed maze programs, pencil, and paper

Have students exchange tapes and try to solve each other's mazes. Ask them to criticize the programs by finding errors and suggesting how to make improvements.

Ask the students what they would like to include in the mazes. For instance, a timed input could be used. Graphics and flashing displays could be inserted. Maybe a counter can be set up so that the player has only a certain number of times that he can run into a blocked exit. The starting square could be randomly generated, as could the winning one.

If time is no object, have the students investigate all of the above. If time is a factor, it would be wise to anticipate student reactions and have the next step well prepared. That is especially true if graphics are to be considered.

Lesson 8: How the Program Works

Time: One session for those students who are interested

Line 160	N = 25 sets the size of the maze at 25 squares.
Line 160	DIM S(N,4) sets up 100 holders for the various destinations. For example, S(5,2) is the number of the square that you wind up in after moving east from square 5.
Lines 170–210	The destinations that are listed in the DATA statements are read into the S array.
Line 220	Sets up the information necessary to put the user into the starting square.
Line 250	There are 26 possible destinations, counting a blocked exit. The 26 numbers listed here are the line numbers at which the square descriptions start.
Line 260	Landing in a death trap or a winning square causes R to be set to zero and ends the game.
Line 510	If the game is still in progress, the direction to move must be obtained.
Line 520	LEFT$(Q$,1) considers just the first letter to the left of Q$. That way, if EAST were entered, Q$ would now be just E. You can omit this part when using BASICs that do not support the LEFT$ function.
Lines 530–560	Converts the four compass directions into their respective codes: N = 1, E = 2, S = 3, W = 4.
Line 570	The program will not accept a letter other than N,E,S, or W to indicate direction.

We've had a lot of fun with this package at practically no cost. It can be used from grade 7 on up and maybe even below grade 7 with proper

guidance. The activity is well suited for group work when only a few computers are available.

While the lessons call for a TRS-80 Model I 16K Level II, the program should work with any computer that supports string variables and two variable numeric arrays.

Program Listing. *Monster's Gold*

Encyclopedia Loader

```
10 REM * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
12 REM * *              MONSTER'S  GOLD                          * *
14 REM * *      BY  ANNE WEISS  AND  STEVE ARMISTEAD             * *
16 REM * *      ST PETER'S HS  NEW BRUNSWICK NJ  08901           * *
18 REM * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
20 REM  **  DESCRIPTION AND DIRECTIONS  **
30 CLS: PRINT "HELLO!  YOU ARE ABOUT TO LEAVE THE 20TH CENTURY AND GO"

40 PRINT "BACK IN TIME TO THE 11TH CENTURY!  HOLD TIGHT!!!"
50 FOR I=1 TO 1000: NEXT I
60 CLS
70 FOR I=1 TO 1500: NEXT I
80 PRINT "YOU MADE IT!  WELCOME TO THE 11TH CENTURY."
90 PRINT "THE YEAR IS 1037.  YOU ARE THORCAZ, THE LEADER OF A BAND OF"

100 PRINT "WARRIORS.  YOUR OBJECTIVE IS TO FIND THE CAVE IN WHICH THE"

110 PRINT "EVIL GIANT, ALTON, HAS HIDDEN HIS TREASURE."
120 PRINT "WHEN THE QUEST STARTS, THE CAVE YOU ARE IN WILL BE DESCRIBE
    D."
130 PRINT: PRINT "G O O D   L U C K  -  YOU'LL NEED IT!!!": PRINT
140 '
150 REM INITIALIZE AND READ DATA
160 LET N=25: DIM S(N,4)
170 FOR R=1 TO N
180   FOR D=1 TO 4
190     READ S(R,D)
200   NEXT D
210 NEXT R
220 LET R=11: LET D=4
230 '
240 REM MAIN LOOP OF PROGRAM
250 ON S(R,D) GOSUB 1000,1010,1020,1030,1040,1050,1060,1070,1080,1090,
    1100,1110,1120,1130,1140,1150,1160,1170,1180,1190,1200,1210,1220,
    1230,1240,1250
260 IF R=0 THEN 1260
500 PRINT: PRINT
510 INPUT "WHICH WAY WOULD YOU LIKE TO MOVE (N,E,S,W) ";Q$
520 PRINT: Q$=LEFT$(Q$,1)
530 IF Q$="N" THEN D=1: GOTO 250
540 IF Q$="E" THEN D=2: GOTO 250
550 IF Q$="S" THEN D=3: GOTO 250
560 IF Q$="W" THEN D=4: GOTO 250
570 PRINT "NO! NO!  PAY ATTENTION NEXT TIME!": GOTO 310
1000 REM SQ. 1
1002 PRINT "YOU ARE NOW IN A LARGE, DARK CAVE.  SUDDENLY YOU HEAR WING
    S    FLAPPING.  OH NO!  IT'S A HARPY!  GET OUT QUICK!"
1005 DATA 11,2,6,26
1009 LET R=1 : RETURN
1010 REM SQ. 2
1012 PRINT "YOU ARE NOW IN A LARGE, COLD CAVE.  THERE ARE ICICLES EVER
    Y-   WHERE.  WATCH OUT!  IT'S A HEAT MONGER!  RUN BEFORE HE FREE
    ZES  YOU TO DEATH!"
1015 DATA 26,3,1,7
1019 LET R=2 : RETURN
1020 REM SQ. 3
1022 PRINT "YOU ARE NOW IN A NARROW CAVERN.  IT IS VERY CHILLY.  SUDDE
    NLY"
1023 PRINT "YOU HEAR WINGS!  GIANT BATS!!  GET OUT QUICK!"
1025 DATA 4,4,7,2
1029 LET R=3 : RETURN
1030 REM SQ. 4
1032 PRINT "IT IS CALM AND QUIET.  YOU AND YOUR GROUP BEGIN TO RELAX.
     OUT  OF NOWHERE A MAN-EATING ROCK MONSTER APPEARS.  HE DEVOURS O
    NE"
1033 PRINT "OF YOUR WARRIORS.  ESCAPE BEFORE HE ATTACKS AGAIN!"
1035 DATA 26,5,9,3
```

```
1039 LET R=4 : RETURN
1040 REM SQ. 5
1042 PRINT" YOU AND YOUR MEN ENTER A CAVE.  SUDDENLY YOU SEE SOMETHING
      MOVE BEHIND A ROCK.  IT'S A GYNOMORPH!  RUN BEFORE SHE KEEPS YOU
      AS  HER PET FOR THE REST OF YOUR LIFE!"
1045 DATA 4,26,10,3
1049 LET R=5:RETURN
1050 REM SQ. 6
1052 PRINT"YOU ENTER A LARGE, HOT CAVE.  THERE IS RED HOT SULFER AND F
     IRE  BURNING ALL AROUND YOU!  GET OUT BEFORE YOU SUFFOCATE OR BUR
     N   TO DEATH!"
1055 DATA 26,7,11,11
1059 LET R=6:RETURN
1060 REM SQ. 7
1062 PRINT "YOU'VE JUST ENTERED A COOL, MISTY CAVE. EVERYONE SUDDENLY
     FEELS
SLEEPY.  LEAVE QUICKLY - YOU'VE JUST ENTERED THE ABODE OF THE
GOD OF SLEEP.  HE MAY MAKE YOU SLEEP FOREVER IF YOU DON'T LEAVE
NOW!"
1065 DATA 2,13,12,6
1069 LET R=7: RETURN
1070 REM SQ 8
1072 PRINT "CAN THIS CAVE BE THE ONE YOU'VE BEEN SEARCHING FOR?  THERE
      ARE  PILES OF GOLD EVERYWHERE.  ONE OF YOUR MEN, FILLED WITH JOY
     ,    JUMPS INTO THE PILE.  OH NO!  IT ISN'T GOLD! IT'S A TRIBE OF
        GOLD COIN-LEACHES.  LEAVE BEFORE THEY ATTACK!"
1075 DATA 3,9,13,7
1079 LET R=8 :RETURN
1080 REM SQ.9
1082 PRINT "THE CAVE YOU'VE JUST ENTERED IS VERY LARGE AND HIGH. THERE
      ARE  ASHES FROM OLD CAMP FIRES.  OUT OF NOWHERE, A CENTAUR RUSHE
     S    OUT AT YOU.  YOU EASILY KILL HIM WITH YOUR MIGHTY SWORD.
LEAVE BEFORE HIS MATE GETS HER REVENGE!"
1085 DATA 3,5,14,8
1089 LET R=9: RETURN
1090 REM SQ. 10
1092 PRINT" YOU SPOT A BEAUTIFUL MAIDEN RUNNING INTO A CAVE.  UPON FOL
     LOW-  ING HER, YOU DISCOVER SHE'S REALLY A WITCH, AND SHE'S GOT Y
     OU   IN HER LAIR.  RUN BEFORE SHE PUTS A SPELL ON YOU!"
1095 DATA 5,15,15,9
1099 LET R=10: RETURN
1100 REM SQ. 11
1102 PRINT" YOU'VE JUST ENTERED A LARGE, WARM CAVE WITH HUGE SILKEN BA
     LLS   ALL OVER.  THEY BEGIN TO CRACK.  OH NO!  THEY'RE THE COCCOO
     NS   OF GIANT MAN-EATING MOTHS!  RUN QUICK!"
1105 DATA 6,12,16,1
1109 LET R=11: RETURN
1110 REM SQ. 12
1112 PRINT "YOU AND YOUR MEN ENTER A LARGE, EMPTY CAVE.  YOU WALK TO T
     HE    CENTER.  THE GROUND SUDDENLY BEGINS TO RUMBLE.  OH NO!  IT'
     S A  GIANT MOLE!  ESCAPE WHILE YOU STILL CAN!
1115 DATA 8,13,16,11
1119 LET R=12 : RETURN
1120 REM SQ. 13
1122 PRINT "YOU'VE FOUND A CAVE IN WHICH YOU'VE DECIDED TO REST. THERE
      IS A POOL NEARBY.  ONE OF YOUR MEN GOES TO GET A DRINK.  HE'S EA
     TEN  BY A SHARKMAN.  GET OUT BEFORE HE DECIDES HE'S STILL HUNGRY!
     "
1125 DATA 26,14,18,12
1129 LET R=13 : RETURN
1130 REM SQ. 14
1132 PRINT "YOU ENTER A SMALL, DAMP CAVE.  THERE IS A BUZZING NOISE E
     VERY- WHERE.  OH NO!  IT'S A NEST OF MAN-EATING ANTS!  RUN QUICK
     LY!"
1135 DATA 10,15,19,13
1139 LET R= 14: RETURN
1140 REM SQ. 15
1142 PRINT " THE CAVE YOU'VE JUST ENTERED IS VERY SMALL AND NARROW.  T
     HERE  ARE SPIDERWEBS ALL OVER.  OH NO!  A GIANT SPIDER!  ESCAPE
          QUICKLY!"
```

Program continued

```
1145 DATA 10,10,20,26
1149 LET R=15 : RETURN
1150  REM SQ 16
1152 PRINT " YOU'VE JUST ENTERED A DARK, DAMP CAVE.  OH NO!  IT'S AN A
     NCIENT EGYPTIAN BURIAL TOMB.  HERE COMES A MUMMY!  GET OUT QUICK!
     "
1155 DATA 11,17,22,26
1159 LET R=16 : RETURN
1160 REM SQ. 17
1162 PRINT "YOU'VE JUST ENTERED A HIGH, ROCKY CAVE.   WATCH OUT!  IT'S
      AN   AVALANCE.  RUN QUICK!"
1165 DATA 26,18,22,6
1169 LET R=17 : RETURN
1170 REM SQ 18 (DEATH TRAP 1)
1172 PRINT "TOO BAD!  YOU'VE ALL JUST FALLEN INTO A HUGE FISSURE AND D
     IED!"
1175 DATA 26,26,26,26
1179 LET R=0: RETURN
1180 REM SQ. 19
1182 PRINT "THE CAVE YOU'VE JUST ENTERED IS VERY SMALL, WARM AND MOIST
     .     SUDDENLY BLOOD BEGINS TO POUR OUT OF THE WALLS!  RUN BEFORE
      YOU ARE DROWNED!"
1185 DATA 1,20,24,18
1189 LET R=19 : RETURN
1190  REM SQ. 20 (DEATH TRAP 2 )
1192 PRINT "YOU'VE STUMBLED UPON A CAVE INHABITED BY CANNIBALS!  OH NO
     !     THEY'RE EATING YOU!  TOO BAD, YOU'RE DEAD NOW!"
1195 DATA 26,26,26,26
1199 LET R=0: RETURN
1200 REM SQ. 21
1202 PRINT " YOU'VE ENTERED A LARGE, HOT, MISTY CAVE.  EVERYONE IS SUD
     DENLY  HAVING BAD FEELINGS ABOUT THE CAVE.  NO WONDER WHY!  IT'S
     THE   HOME OF rICZAZ, THE EVIL MAGICIAN.  RUN BEFORE HE FINDS YOU
     !"
1205 DATA 16,22,22,26
1209 LET R=21 : RETURN
1210 REM SQ. 22
1212 PRINT" YOU'VE ENTERED A LARGE, HIGH CAVE.  YOU SPY ANOTHER GROUP.
           THERE IS A FIGHT AND YOU'RE OVERPOWERED.  ESCAPE BEFORE YOU
          ARE KILLED!"
1215 DATA 17,23,23,21
1219 LET R= 22 : RETURN
1220 REM SQ. 23
1222 PRINT "YOU'VE ENTERED A LARGE, WARM, MOIST CAVE.  THERE ARE PLANT
     S ALL OVER.  YOU GO TO EAT THEM, BUT THEY EAT YOU FIRST!  RUN IF
     YOU  STILL CAN!"
1225 DATA 18,26,25,22
1229 LET R=23 : RETURN
1230 REM SQ. 24
1232 PRINT "CONGRATULATIONS!   YOU'VE FOUND THE CAVE WITH THE TREASURE
     S!"
1235 DATA 26,26,26,26
1239 LET R=0: RETURN
1240 REM SQ. 25
1242 PRINT "YOU'VE ENTERED A LARGE CAVE WITH TREASURES ALL OVER.  IT'S
      THE  ABODE OF THE GOD OF THE UNDERWORLD.  ESCAPE BEFORE HE KEEPS
      YOU FOR GOOD!"
1245 DATA 20,1,21,26
1249 LET R=25 : RETURN
1250 REM BLOCK
1252 PRINT "OH NO! THE WAY IS BLOCKED! GO ANOTHER WAY BEFORE IT'S TOO
     LATE!"
1259 RETURN
1260 REM GAME IS OVER
1270 PRINT: PRINT: PRINT
1280 FOR I=1 TO 1000: NEXT I
1290 INPUT "WOULD YOU LIKE TO PLAY AGAIN (Y OR N) ";Q$
1300 IF LEFT$(Q$,1)="Y" THEN RUN
1310 END
```

Getting Your Bearings

by Gary V. Small

In the world of computer games and simulations, one of the basic problems encountered by the programmer is finding the bearing of a known point, relative to another point, referenced to a 360 degree compass. Navigation and radar computer systems are dedicated to similar problems in ships, aircraft, and ground radar installations. Finding a bearing between two points is a relatively simple exercise in trigonometry. Due to the use of a compass as a reference system, and the limitations of inverse trig functions, some care must be used in setting up the problem for use on a computer system.

As a first step, we will define an X-Y world with two points in it, as shown in Figure 1. In an actual application, this might represent an airplane and an airport, two airplanes, or most any other kind of object. For the purposes of calculating a bearing, you are looking at a frozen moment of time, so any movement of the objects does not come into play. You simply have two points, with absolute positions given by their X- and Y-coordinates.

You must now differentiate between the two points and label them A and B. You must select one of these points as the reference point for calculating the bearing. With point A as the reference point, the other point now becomes the "target" point to which the bearing is calculated. If point A was a ship trying to reach point B, it would assume a compass course equal to the bearing to point B.

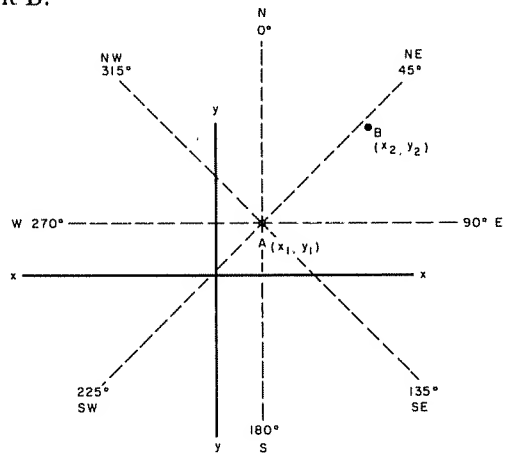


Figure 1

Look at Figure 1 again and notice the compass reference points. In working this problem, you are actually dealing with two coordinate systems. There is the absolute X-Y world, and a compass-referenced world which is always centered on the reference point. The concepts of bearing and range are relative to the two points and are constantly centered on the reference point. Notice in Figure 1 that the angle increases in a clockwise direction, and that zero degrees is vertically oriented. This is the opposite of the usual angular orientation used in working with points in an X-Y system.

Figure 2 shows the triangular construct used to find the range and bearing to point B with respect to point A. The length of side XX is given by the formula $XX = (X_2 - X_1)$; the length of YY is given by the formula $YY = (Y_2 - Y_1)$. Note the order of the subtraction, as this order will give the appropriate sign for XX and YY to be used later. The long side of the triangle, which is also the range between the two points, is given by the formula $R = (XX^2 + YY^2)^{1/2}$.

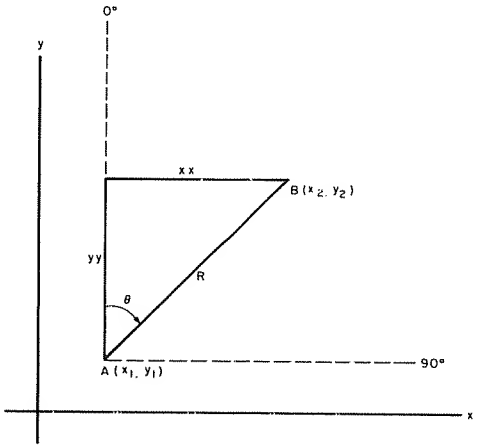


Figure 2

As shown in the figure, the angle you are interested in is θ (theta). This is the bearing to the target point from point A. Since you have constructed a right triangle, the sine of θ is defined as the opposite side/hypotenuse, or XX/R. The cosine is defined as the adjacent side/hypotenuse, or YY/R. Note that you have effectively reversed the roles of the X- and Y-axes, when compared to standard X-Y referenced angular constructs. This is done to accommodate the compass reference which has been superimposed on the X-Y world.

Now that you know the sine and cosine, you should be able to look up the corresponding angle in a table of inverse trig functions. You can look up an angle within the range of 0–90 degrees, but after that, difficulties arise. Look at Figure 3. This shows the sign of the trig function in the various portions of the circle. The determination of the bearing requires

that you know which quadrant you are in. If you look a little further, you find that the TRS-80 has only one inverse trig function, the arctangent (BASIC keyword ATN). You must use more complex techniques to solve for the bearing on a computer.

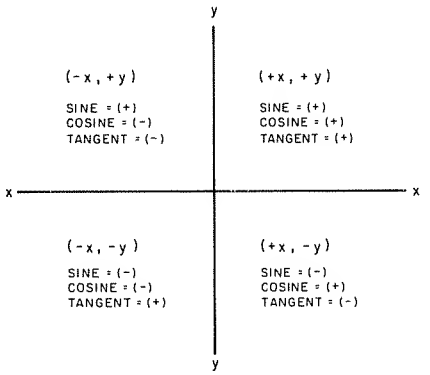


Figure 3

The tangent of an angle is defined as the opposite side/adjacent side, or XX/YY. The signs of XX and YY determine which quadrant the angle is in, so define the function TT as the absolute value of (XX/YY), and thus confine it to the 0–90 degree range. Note that if YY is zero, a division by zero error occurs. You must flag any condition in which YY is zero. This occurs only if the target point has the same Y-coordinate as the reference point, which means the target is directly east or west of the reference point, at a bearing of 90 or 270 degrees. If XX is positive the bearing is 90 degrees, if XX is negative the bearing is 270 degrees.

The Bearings program, shown in the Program Listing, returns the range and bearing when you enter the X- and Y-coordinates of a reference point and a target point.

The program first uses the coordinate values of the two points to calculate the values XX, YY, and R, the range. To eliminate any division by zero problems, the program then checks for XX and YY values that would occur at the 0, 90, 180, and 270 degree points. If the bearing is not one of these points, then a value is calculated for TT, the tangent, and for C, a degrees/radians conversion factor. Lines 100–130 generate the bearing within the four different quadrants. The signs of XX and YY are used to determine the quadrant. The arctangent value, which is in the 0–90 degree range, is added to an offset value (in the third quadrant) or subtracted from an offset value (second and fourth quadrants). The subtraction is necessary in those quadrants due to the nature of the arctangent function, which increases in the wrong direction in those quadrants. You now have a working bearings routine. You can convert this program to a subroutine by

deleting the INPUT and PRINT statements, replacing the END statement with a RETURN, and providing the X- and Y-coordinates of the two points from the main program.

```
10 REM X1,Y1 IS REFERENCE POINT , X2,Y2 IS TARGET , R IS RANGE , AN IS
     BEARING
20 INPUT"INPUT X1,Y1";X1,Y1 : INPUT"INPUT X2,Y2";X2,Y2
30 XX=(X2-X1) : YY=(Y2-Y1) : R=SQR(XX[2 + YY[2)
40 IF YY=0 AND XX>=0 THEN AN=90 : GOTO 140
50 IF YY=0 AND XX<0 THEN AN=270 : GOTO 140
60 IF XX=0 AND YY>=0 THEN AN=0 : GOTO 140
70 IF XX=0 AND YY<0 THEN AN=180 : GOTO 140
80 TT=ABS(XX)/ABS(YY)
90 C=(2*3.1415926)/360
100 IF YY>0 AND XX>0 THEN AN=ATN(TT)/C
110 IF YY<0 AND XX>0 THEN AN=180-ATN(TT)/C:REM IN 2ND QUADRANT,    ATN
      VALUE STARTS AT 90 DEG AND DECREASES
120 IF YY<0 AND XX<0 THEN AN=180+ATN(TT)/C
130 IF YY>0 AND XX<0 THEN AN=360-ATN(TT)/C :REM IN 4TH QUADRANT,  ATN
     VALUE STARTS AT 90 DEG AND DECREASES
140 PRINT "RANGE =";R,"BEARING=";AN
150 END
```

GAMES

Left/Right for the Color Computer
Munch

Left/Right for the Color Computer

by Robert Toscani

An article in a recent issue of a computer magazine described a game called Left/Right. A box of some color appears on the screen along with one colored bar on each side. Depending on the background screen color, the player presses the appropriate joystick button to indicate which bar matches the box's color.

Keeping in mind the details of the game, I wrote a program for the Color Computer. (See Program Listing 1.) With a little work, I came up with a working program. But that old demon, the desire for improvement, sneaked in, and the program changed as features were altered and added. I stayed with commands that would run on a 4K machine, but by the time I had something I was satisfied with, it needed over 5000 bytes to run. To run the program on a 4K machine, enter Program Listing 2.

The Program

Line 10 is for the winners list, deleted in the 4K program. Users with 4K should skip to line 450. Lines 20–150 need no explanation. Lines 160–190 set up an input from the joystick buttons. To use keyboard inputs, set up an INKEY$ loop. These lines could have been combined, and I suggest you do so. The program is written in this form for clarity.

Lines 200–440 set up a demonstration of what the game looks like and, again, input from the buttons. Line 450 sets the starting score at zero. Lines 460–560 establish how the game will be played.

Line 600 begins the game and makes the screen either black or white. Lines 640–650 decide the left and right bar colors, with line 660 making sure they are not the same. Some of the colors are similar, making deciding difficult.

Lines 670–690 determine if the left or right bar will be the same as the box. Lines 700–730 SET the bars, while lines 740–790 pick the box's location and SET it. Lines 800–830 print the timer and the score, subtract one unit from the time, and end the game if time runs out.

Lines 840–920 provide joystick input and tell the conditions for a right or wrong response. It is set up to be used when the screen is black or white, even though the buttons are reversed.

When the screen is black, press the button corresponding with the matching bar. If the left bar matches the box, press the left button. If the screen is white, do the opposite. In the example, with a white screen, you would press the right button.

If you do not have a joystick, write an INKEY$ loop for lines 850, 870, 890, 910 and delete the other lines. Lines 940–950 send the box jumping. I originally tried using the RESET statement; it worked for the black screen but did not work properly with the white. It reset the box to black instead of white. I had to use SET and give the color code to get it to work.

Line 960 sets up the loop to keep the clock running. Lines 970–1000 set the screen white, pick the bar colors, and make sure they are not the same or white. Lines 1010–1030 reverse the bar and box color relationship and set up for the black screen, permitting lines 840–920 to be used for both black and white. Lines 1050–1150 are the right and wrong responses. Line 1080 ends the game if you achieved your score. Lines 1160–1250 are the RESET lines to make the box jump around.

Line 1310 determines if you are eligible for the winners list. You are eligible for the winners list if: You have specified a time limit of 300 or less, you have a score of 50 or more, you have specified that wrong answers count against you, and that the box moves around. You can change this if you wish.

Anyone making it to the winner is rewarded with fireworks going off and a chorus of *Stars and Stripes Forever*. The graphics are poor and should be deleted by 4K users. I used one sound per line; however, using multiple lines will save a little memory. Those of you with Extended BASIC should use the PLAY option and really shorten the program. It will also sound better. You can easily transpose from one code to the other using the textbooks. One caution—the textbook is wrong when it says the computer automatically uses octave 2 if not specified. It actually uses octave 3. The end lines are self explanatory and optional.

Program Listing 1. *Left/Right*

```
10 G=1
20 CLS
30 PRINT "A COLORED BOX WILL APPEAR ON "
40 PRINT "THE SCREEN, SOMEWHERE. TWO BARS"
50 PRINT "WILL BE ON THE BOTTOM. YOU MUST"
60 PRINT "DETERMINE IF THE LEFT OR RIGHT"
70 PRINT "BAR COLOR MATCHS THE BOX COLOR"
80 PRINT "AND PRESS THE CORRESPONDING"
90 PRINT "BUTTON. DO THIS IF THE SCREEN"
100 PRINT "IS BLACK. IF IT IS WHITE, YOU"
110 PRINT "MUST PRESS THE OTHER BUTTON."
120 PRINT "TO SEE WHAT THIS WOULD LOOK"
130 PRINT "LIKE, PRESS THE LEFT HAND   "
140 PRINT "BUTTON. IF YOU WANT TO GO ON TO"
150 PRINT "THE GAME, PRESS THE RIGHT ONE."
160 A=PEEK(65280)
170 IF A=125 OR A=253 THEN 200
180 IF A=126 OR A=254 THEN 450
190 GOTO 160
200 FOR M=1 TO 300:NEXT M
205 C=1
210 CLS(0)
220 SET(30,15,1)
230 SET(30,16,1)
240 SET(31,15,1)
250 SET(31,16,1)
260 FOR B=12 TO 20
270 SET(B,30,1)
280 SET(B+32,30,3)
290 NEXT B
300 IF C=2 THEN 370
310 PRINT "HERE YOU WOULD PRESS THE LEFT"
320 PRINT "BUTTON."
330 C=C+1
340 FOR M=1 TO 1000:NEXT M
350 CLS(5)
360 GOTO 220
370 PRINT "BUT HERE, YOU WOULD PRESS THE"
380 PRINT "RIGHT. IF YOU'RE READY TO GO,"
390 PRINT "PRESS THE RIGHT BUTTON. IF NOT"
400 PRINT "PRESS THE LEFT."
410 A=PEEK(65280)
420 IF A=125 OR A=253 THEN 20
430 IF A=126 OR A=254 THEN 450
440 GOTO 410
450 T(G)=0
460 CLS
470 FOR M=1 TO 300:NEXT M
480 INPUT "TIME? TO START, TRY 1000";I(G)
490 E=I(G)
500 PRINT "ENTER A GOAL SCORE. IF YOU"
510 INPUT "DON'T WANT ONE, PRESS 0";F
520 PRINT "WHAT ABOUT HAVING THE BOX MOVE"
530 INPUT "AROUND? Y/N";G$
540 PRINT"IF YOU HAVE A GOAL SCORE, DO"
550 PRINT "YOU WANT WRONG ANSWERS TO COUNT"
560 INPUT "AGAINST YOU? Y/N";I$
570 INPUT "WHAT'S YOUR NAME";K$(G)
580 PRINT "OK, "K$(G)" GET SET, WE ARE JUST ABOUT TO BEGIN."
590 FOR M=1 TO 1000:NEXT M
600 N=RND(2)
610 IF N=1 THEN 630
620 IF N=2 THEN 970
630 CLS(0)
640 J=RND(8)
650 L=RND(8)
660 IF J=L THEN 640
```

Program continued

```
670 S=RND(2)
680 IF S=1 THEN K=J
690 IF S=2 THEN K=L
700 FOR B=12 TO 20
710 SET(B,30,J)
720 SET(B+29,30,L)
730 NEXT B
740 P=RND(58)
750 Q=RND(27)
760 SET(P,Q,K)
770 SET(P+1,Q,K)
780 SET(P,Q+1,K)
790 SET(P+1,Q+1,K)
800 PRINT @ 480,E;
810 PRINT @ 506,T(G);
820 E=E-1
830 IF E=0 THEN 1330
840 A=PEEK(65280)
850 IF S=1 AND A=253 THEN 1050
860 IF S=1 AND A=125 THEN 1050
870 IF S=1 AND A=254 THEN 1110
880 IF S=1 AND A=126 THEN 1110
890 IF S=2 AND A=253 THEN 1110
900 IF S=2 AND A=125 THEN 1110
910 IF S=2 AND A=254 THEN 1050
920 IF S=2 AND A=126 THEN 1050
930 SOUND 130,1
940 IF G$="Y" AND N=1 THEN 1160
950 IF G$="Y" AND N=2 THEN 1210
960 GOTO 800
970 CLS(5)
980 J=RND(8)
990 L=RND(8)
1000 IF J=5 OR L=5 OR J=L THEN 980
1010 S=RND(2)
1020 IF S=1 THEN K=L
1030 IF S=2 THEN K=J
1040 GOTO 700
1050 PRINT @ 493,"RIGHT";
1060 T(G)=T(G)+1
1070 IF E=0 THEN 1330
1080 IF T(G)=F AND F>0 THEN 1260
1090 FOR M=1 TO 500:NEXT M
1100 GOTO 600
1110 PRINT @ 493,"WRONG";
1120 IF I$="Y" THEN T(G)=T(G)-1
1130 IF E=0 THEN 1330
1140 FOR M=1 TO 300:NEXT M
1150 GOTO 600
1160 RESET(P,Q)
1170 RESET(P+1,Q)
1180 RESET(P,Q+1)
1190 RESET(P+1,Q+1)
1200 GOTO 740
1210 SET(P,Q,5)
1220 SET(P+1,Q,5)
1230 SET(P,Q+1,5)
1240 SET(P+1,Q+1,5)
1250 GOTO 740
1260 FOR M=1 TO 300:NEXTM
1270 CLS
1280 PRINT"YOU WIN,"K$(G)"!"
1290 PRINT "YOUR SCORE IS "T(G)
1300 FOR M=1 TO 1000:NEXT M
1310 IF I(G)<=300 AND T(G)=>50 AND G$="Y" AND I$="Y" THEN 1560
1320 GOTO 1460
1330 FOR M=1 TO 300:NEXTM
1340 CLS
1350 PRINT "TIME'S UP, "K$(G)
1360 IF F>0 THEN 1410
```

```
1370 PRINT "YOUR SCORE IS "T(G)
1380 PRINT "NOT TOO BAD."
1390 FOR M=1 TO 1000:NEXT M
1400 GOTO 1460
1410 PRINT "SORRY, YOU LOSE."
1420 PRINT "YOUR SCORE WAS ONLY "T(G)"AND YOUR GOAL WAS "F"."
1430 PRINT "BETTER LUCK NEXT TIME, "K$(G)
1440 FOR M=1 TO 1000:NEXTM
1460 CLS
1470 PRINT "DO YOU WANT TO GO AGAIN?"
1480 PRINT "IF YOU WANT TO START FROM THE"
1490 PRINT "BEGINNING, PRESS 1. IF YOU WANT"
1500 PRINT "TO GO BACK TO THE MENU, PRESS"
1510 PRINT"2.  IF YOU JUST WANT TO END IT,"
1520 INPUT "PRESS 3.";Z
1530 IF Z=1 THEN 20
1540 IF Z=2 THEN 450
1550 IF Z=3 THEN END
1560 CLS
1570 PRINT "SINCE YOU HAVE DONE SO WELL,"
1580 PRINT "WE ARE PUTTING YOUR NAME,TIME"
1590 PRINT "AND SCORE IN OUR CHANPION LIST."
1600 FOR M=1 TO 500:NEXT M
1610 CLS
1620 FOR D=1 TO G
1630 PRINT K$(D)        I(D)  T(D)
1640 NEXT D
1650 G=G+1
1660 IF G=13 THEN 1690
1670 INPUT "DO YOU WANT TO END THE RUN Y/N";Y$
1680 IF Y$="N" THEN 1460
1690 CLS: IF G=13 THEN 1710
1700 IF Y$="Y" THEN 1720
1710 PRINT "OUT OF SPACE, PEOPLE."
1720 PRINT "OK, THEN THAT'S THE END OF THIS"
1730 PRINT "RUN. BUT WE HAVE A LITTLE PRIZE"
1740 PRINT "FOR DOING SO WELL."
1750 FOR M=1 TO 1000:NEXT M
1760 GOSUB 2600
1770 SOUND 165,8
1780 SOUND 165,8
1790 SOUND 153,4
1800 SOUND 147,4
1810 SOUND 147,8
1820 GOSUB 2770
1830 GOSUB 2860
1840 SOUND 147,22
1850 H=4:GOSUB 2790
1860 GOSUB 2860
1870 SOUND 147,8
1880 H=5:GOSUB 2790
1890 H=6:GOSUB 2790
1900 GOSUB 2860
1910 SOUND 165,8
1920 SOUND 147,6
1930 SOUND 165,2
1940 SOUND 153,16
1950 SOUND 133,12
1960 H=8:GOSUB 2790
1970 H=10:GOSUB 2790
1980 SOUND 133,4
1990 SOUND 133,8
2000 GOSUB 2890
2010 SOUND 133,8
2020 GOSUB 2890
2030 SOUND 153,22
2040 H=12:GOSUB 2790
2050 SOUND 143,4
2060 SOUND 133,4
2070 SOUND147,4
```

Program continued

```
2080 SOUND 165,12
2090 SOUND 176,12
2100 SOUND 176,4
2110 SOUND 133,22
2120 H=15:GOSUB 2790
2130 SOUND 165,8
2140 SOUND 165,8
2150 SOUND 153,4
2160 SOUND 147,4
2170 SOUND 147,8
2180 H=20:GOSUB 2790
2190 GOSUB 2860
2200 SOUND 147,22
2210 H=21:GOSUB 2790
2220 GOSUB 2860
2230 SOUND 147,8
2240 GOSUB 2860
2250 SOUND 153,4
2260 SOUND 147,4
2270 SOUND 133,6
2280 SOUND 108,2
2290 SOUND 133,16
2300 SOUND 117,12
2310 GOSUB 2600
2320 SOUND 117,4
2330 SOUND 117,8
2340 SOUND 108,4
2350 SOUND 117,4
2360 SOUND 140,8
2370 GOSUB 2770
2380 SOUND 133,4
2390 SOUND 117,4
2400 SOUND 189,20
2410 SOUND 117,4
2420 SOUND 133,4
2430 SOUND 147,4
2440 SOUND 165,2
2450 H=4:GOSUB 2790
2460 SOUND 117,4
2470 SOUND 133,4
2480 SOUND 147,4
2490 SOUND 165,2
2500 H=6:GOSUB 2790
2510 SOUND 69,4
2520 SOUND 89,4
2530 SOUND 147,4
2540 SOUND 133,16
2550 SOUND 117,4
2560 FOR H=7 TO 22
2570 GOSUB 2790
2580 NEXT H
2590 GOTO 2920
2600 CLS(0)
2610 FOR D=1 TO 23
2620 SET(8,23-D,1)
2630 SET(7,24-D,1)
2640 SET(8,24-D,1)
2650 SET(9,24-D,1)
2660 SET(7,25-D,1)
2670 SET(8,25-D,1)
2680 SET(9,25-D,1)
2690 SET(7,26-D,4)
2700 SET(8,26-D,4)
2710 SET(9,26-D,4)
2720 SET(8,27-D,4)
2730 CLS(0)
2740 NEXT D
2750 CLS(RND(5))
2760 RETURN
2770 CLS(0)
```

```
2780 H=2
2790 V=RND(5):D=RND(5):U=RND(5)
2800 SET(D+U,U,V)
2810 SET(H+D+U,U,V)
2820 SET(H+D+U,D+U,V)
2830 SET(H+D+U+V,D+U,V)
2840 SET(H+D+U+V,D+U+H,V)
2850 RETURN
2860 SOUND 140,4
2870 SOUND 147,4
2880 RETURN
2890 SOUND 125,4
2900 SOUND 133,4
2910 RETURN
2920 FOR M=1 TO 1000:NEXT M
2930 CLS
2940 PRINT "PRETTY BAD,HUH?"
2950 PRINT "WELL, WE'RE TIRED TOO. IT TAKES"
2960 PRINT "A LOT OF RUNNING AROUND INSIDE"
2970 PRINT "THIS LITTLE COMPUTER TO KEEP"
2980 PRINT "IT GOING. SO TAKE A BREAK, "
2990 PRINT "STRETCH YOUR LIMBS AND LET US"
3000 PRINT "RELAX A LITTLE. BYE NOW."
```

Program Listing 2. *4K version*

```
10 T=0
20 CLS
30 FORM=1TO300:NEXTM
40 INPUT"TIME? TO START, TRY 1000";E
50 PRINT"ENTER A GOAL SCORE. IF YOU"
60 INPUT"DON'T WANT ONE, PRESS 0";F
70 PRINT"WHAT ABOUT HAVING THE BOX MOVE"
80 INPUT"AROUND? Y/N";G$
90 PRINT"IF YOU HAVE A GOAL SCORE, DO"
100 PRINT"YOU WANT WRONG ANSWERS TO COUNT"
110 INPUT"AGAINST YOU? Y/N";I$
120 INPUT"WHAT'S YOUR NAME";K$
130 PRINT"OK, "K$" GET SET, WE ARE JUST ABOUT TO BEGIN."
140 FORM=1TO1000:NEXTM
150 N=RND(2)
160 IFN=1THEN170ELSEIFN=2THEN410
170 CLS(0)
180 J=RND(8):L=RND(8)
190 IF J=L THEN180
200 S=RND(2)
210 IFS=1THENK=J
220 IFS=2THENK=L
230 FORB=12TO20
240 SET(B,30,J):SET(B+29,30,L):NEXTB
250 P=RND(58):Q=RND(27)
260 SET(P,Q,K):SET(P+1,Q,K):SET(P,Q+1,K):SET(P+1,Q+1,K)
270 PRINT@480,E;:PRINT@506,T;
280 E=E-1:IFE=0THEN610
290 A=PEEK(65280)
300 IFS=1ANDA=253THEN480
310 IFS=1ANDA=125THEN480
320 IFS=1ANDA=254THEN520
330 IFS=1ANDA=126THEN520
340 IFS=2ANDA=253THEN520
350 IFS=2ANDA=125THEN520
360 IFS=2ANDA=254THEN 480
370 IFS=2ANDA=126THEN480
380 SOUND130,1
390 IFG$="Y" ANDN=1THEN560ELSEIF G$="Y"ANDN=2THEN 570
400 GOTO270
410 CLS(5)
```

Program continued

```
420 J=RND(8):L=RND(8)
430 IF J=5 OR L=5 OR J=L THEN 420
440 S=RND(2)
450 IFS=1THENK=L
460 IFS=2THENK=J
470 GOTO230
480 PRINT@493,"RIGHT";
490 T=T+1:IFE=0THEN610
500 IF T=F AND F>0 THEN 580
510 FORM=1TO300:NEXTM:GOTO150
520 PRINT@493,"WRONG";
530 IFI$="Y"THENT=T-1
540 IFE=0THEN610
550 FORM=1TO300:NEXTM:GOTO150
560 RESET(P,Q):RESET(P+1,Q):RESET(P,Q+1):RESET(P+1,Q+1):GOTO250
570 SET(P,Q,5):SET(P+1,Q,5):SET(P,Q+1,5):SET(P+1,Q+1,5):GOTO250
580 FORM=1TO300:NEXTM:CLS
590 PRINT"YOU WIN, "K$"!":PRINT"YOUR SCORE IS "T
600 FORM=1TO1000:NEXTM:GOTO700
610 FORM=1TO300:NEXTM:CLS
620 PRINT"TIME'S UP, "K$
630 IFF>0THEN660
640 PRINT"YOUR SCORE IS "T:PRINT"NOT TOO BAD."
650 FORM=1TO1000:NEXTM:GOTO700
660 PRINT"SORRY, YOU LOSE."
670 PRINT"YOUR SCORE WAS ONLY "T"AND YOUR GOAL WAS "F"."
680 PRINT"BETTER LUCK NEXT TIME, "K$
690 FORM=1TO1000:NEXTM
700 CLS
710 PRINT"DO YOU WANT TO GO AGAIN,":PRINT"IF YOU DO, PRESS 1. IF YOU J
UST":PRINT"WANT TO END PRESS ANY KEY."
720 INPUT Z
730 IFZ=1THEN10
740 PRINT"OK, THAT'S IT EXCEPT FOR THIS":PRINT"LITTLE REWARD."
750 FORM=1TO1000:NEXTM
760 SOUND165,8:SOUND165,8:SOUND153,4
770 SOUND147,4:SOUND147,8:GOSUB1000
780 SOUND147,22:GOSUB1000:SOUND147,8
790 GOSUB1000:SOUND165,8:SOUND147,6
800 SOUND165,2:SOUND153,16:SOUND133,12
810 SOUND133,4:SOUND133,8:GOSUB1010
820 SOUND133,8:GOSUB1010
830 SOUND153,22:SOUND143,4:SOUND133,4
840 SOUND147,4:SOUND165,12:SOUND176,12
850 SOUND176,4:SOUND133,22:SOUND165,8
860 SOUND165,8:SOUND153,4
870 SOUND147,4:SOUND147,8:GOSUB1000
880 SOUND147,22:GOSUB1000:SOUND147,8
890 GOSUB1000:SOUND153,4:SOUND147,4
900 SOUND133,6:SOUND108,2:SOUND133,16
910 SOUND117,12:SOUND117,4:SOUND117,8
920 SOUND108,4:SOUND117,4:SOUND140,8
930 SOUND133,4:SOUND117,4:SOUND189,20
940 SOUND117,4:SOUND133,4
950 SOUND147,4:SOUND165,2:SOUND117,4
960 SOUND133,4:SOUND147,4:SOUND165,2
970 SOUND69,4:SOUND89,4:SOUND147,4
980 SOUND133,16:SOUND117,4
990 END
1000 SOUND140,4:SOUND147,4:RETURN
1010 SOUND125,4:SOUND133,4:RETURN
```

Munch

by John Corbani

You have probably paid your dues at the local Pac-Man™ game and wondered if a similar game could run on your TRS-80. You can buy a machine-language version that is amazingly close. This variation I call Munch can teach you some interesting things about BASIC and test your strategy as well as your reflexes.

Munch (see Program Listing) is a chase played in a maze. You, Big Star, are chased through the maze by your son, Little Star. The maze is filled with vitamins that help you run faster and give you game points. The object of the game is to eat all of the vitamins and then hide in the middle of the maze and watch Little Star run around trying to find you. The four keyboard arrow keys control your direction of travel. If you are out in the maze and you don't move, you have from one to 15 seconds before the little guy catches you. Your vitamin count is displayed in the lower left corner of the screen, and the highest score of the session is displayed in the lower right corner.

The program defines the maze in a unique way. The displayed maze consists of full white character cells (CHR$(191)). The vitamins are asterisks. A few periods are thrown in for flavor. Table 1 shows the program variables. The whole thing is held in string array D$(15). The usual technique for generating strings containing graphics characters is to use data statements to hold the numeric codes. This is tedious to program and difficult to modify. There is a better way. Type the pattern on the screen using convenient alphanumeric characters and then write a line of code that replaces selected characters with the desired graphics ones. The program gets double duty out of the translation routine by using it as a timing loop for the game introduction. The total code required to define the screen is about one fourth of that required by numeric data statements, and your BASIC editor can change the maze in nothing flat. Note that the number of characters is limited to 60 per line in this program so that the image is identical on the screen and the printer. You can get full 64-character lines if you let the printer run past 64 characters per line. Remember that the last line printed on the screen cannot exceed 63 characters or the screen scrolls.

Little Star is represented by a plus sign. Big Star is one of six two-character strings, depending on whether he is standing still (F$), moving left (L$), moving right (R$), moving up (U$), moving down (D$), or temporarily not there (E$). If Big Star is standing still, he blinks. If he is moving, his mouth is munching all of the time in the hope that some vitamins will be there.

Numeric Variables

A	=	Temporary loop variable
B	=	Temporary loop variable
HS	=	High score of session
I	=	Keyboard input variable
LA	=	Little Star address
LP	=	Little Star position
LS	=	Little Star step
M	=	Blink counter
P	=	Big Star position
P1	=	Next character
P2	=	Next character + 1
SC	=	Game score
T	=	Temporary Big Star address
TG	=	Toggle flag
X	=	Big Star's X position

String Variables

D$(15)	=	Maze array
D$	=	Mouth down
E$	=	Empty Big Star
F$	=	Full Big Star
I$	=	Temporary string
IN$	=	Temporary string
L$	=	Mouth left
P$	=	Mouth string
R$	=	Mouth right
U$	=	Mouth up
Z$	=	Dummy string

Table 1. *Program variables*

Little Star runs like mad in straight lines, looking neither left nor right until he runs into a wall. Then he jumps up, looks around to see where Big Star is, and heads down the nearest path, usually heading in Big Star's direction. The process repeats until Little Star finds Big Star. Big Star's position is controlled by the four arrow keys. The PEEK function determines which key or keys you press. The program checks to see what lies in the direction you specify. If CHR$(191) is there, Big Star cannot move but he tries to bite the wall. If a blank space or an asterisk is there, movement is allowed. Any vitamins eaten are totaled. If the two stars run into each other, a yell of delight occurs, and the program asks if you would like to run again. Press ENTER to continue. If you get all of the vitamins, press ENTER to start over. There are 200 vitamins in the maze, and a score of 25 is typical for a beginner.

The reaction time of the program is not at all typical of most BASIC games. A walk through the code should explain how the speed was obtained. First, all variables are set to integer, string space is cleared, and the screen array is defined. Then, string variables are defined, and the array characters are changed as required. The string functions are not the fastest way to perform the translation, but they are straightforward, and the countdown properly sets up new players. Big Star's characters are defined, and all important game variables are initialized. Subsequent games begin at line 120. The program then prints the screen from the D$ array.

The program then drops through the Big Star blink routine and jumps to see how Little Star is doing. Little Star is erased, and the program looks ahead one step. If the next character position is not 191, the program checks to see if Big Star has been caught. If he has, the program prints the GOTCHA message, checks for a new high score, and prompts for a new game. If the way is clear, position and address counters are updated, Little Star is POKEd into position, and the program looks for keystrokes.

If Little Star is against a wall (191), the code at 230 figures which of two directions heads towards Big Star and sets LS (Little Step) as required. Right angle turns are tried first, and line 240 makes sure that things don't get stuck when a dead end is encountered. The program loops back to 220, and Little Star moves off in the new direction.

Line 160 is the keyboard scanning section. The four arrow keys and the ENTER key are read by setting I equal to PEEK(14400). If I is 0, then no key is down. The program goes through the blink routine, lets Little Star move, and looks at the keys again. If I equals 1, then the ENTER key is down, and the game player wants out. Head for the end of the program. If I is greater than 1, the program figures that an arrow key is down and sets T (Temporary) equal to P (Big Star's position). The code strips each key in turn, sets the proper character for Big Star, looks ahead one step, and updates P and SC (the score) as appropriate. Big Star is then erased at his old position (T), the program checks the toggle flag (TG), prints either a cross or an open mouth, and complements the toggle flag. Control falls down, checks on Little Star, and loops back to the keyboard scan. If you find that Little Star is too fast, loop back to 160 at the end of 210. Little Star then moves half as fast. I apologize for the condensed and virtually unreadable code in the key parser, but that is the price you have to pay for speed. Adding the line feed and the spaces after each line helps readability a little, but the slower speed is noticeable. Take them out when you get better at the game.

By now you should have a good understanding of the program flow and can modify it with confidence. You can change the characters or the step size. Save your program before you run it, whatever you do. Any program that POKEs into memory at high speed can eat itself up if there is a small error in coding.

Program Listing

Encyclopedia Loader

```
10 '                    MUNCH
              TAPE #15 -1, 5-45; -2, 5-45 "M"
                   BY JOHN CORBANI
                        1/19/82
 .
20 CLEAR 5000: DEFINT A-Z: DIM D$(16)
30 D$(0)=
"??????????????????????????????????????????????????????????????????"
   :D$(1)=
"?? * * * * * * * * * *?  MUNCH  ?* * * * * * * * * * * * *?"
   :D$(2)=
"?? *????????????????????? *???????????* ????????????????????? *?"
40 D$(3)=
"?? * * * * * * * * * * * * * * * * * * * * * * * * * *???? *?"
   :D$(4)=
"?? *?????????????????????????????? *?????????????????? *???? *?"
   :D$(5)=
"?? * * * * * * * * *?????????? *?? * * * * * * * * * * * * *?"
50 D$(6)=
"??????????????????????? *??????????..?? *?????????????????????? *?"
   :D$(7)=
"?? * * * * * * * * * *??          ?? * * * * * * * * * * * *?"
   :D$(8)=
"?? *??????? *???????? *??          ?? *?????????????????????????"

60 D$(9)=
"?? *??????? *???????? *??..?????????? * * * * * * * * * * * *?"
   :D$(10)=
"?? *??????? *???????? * * * * * * * * *?????????????????????? *?"
   :D$(11)=
"?? *??????? *???????? *?????????????? *???? * * * * * * * * *?"
70 D$(12)=
"?? *??????? * * * * * * * * * * * * * * * * *?????????????? *?"
   :D$(13)=
"?? * * * * *?????????????????????????????????? * * * * * * *?"
   :D$(14)=
"??????????? * * * * * * * * * * * * * * * * * *????????????????"
80 D$(15)=
"SCORE    ?????????????????????????????????????????HIGH "
90 CLS: PRINT @ 90, "MUNCH":
   FOR A=0 TO 15: PRINT @ 0, CHR$(30) 15-A;: IN$="":
   FOR B=1 TO 61: I$=MID$(D$(A),B,1):
   IF I$="?" THEN I$=CHR$(191)
100 IN$=IN$+I$: NEXT: D$(A)=IN$:
    IF A=1 PRINT @ 219, "BY":
    PRINT TAB(22) "JOHN CORBANI": ELSE
    IF A=4 PRINT @ 398,
    "THE LITTLE STAR IS AFTER YOU": ELSE
    IF A=9 PRINT @ 532, "GET READY TO RUN": ELSE
    IF A=14 PRINT @ 666, "NOW "
105 NEXT
110 U$=CHR$(165)+CHR$(154):
    D$=CHR$(150)+CHR$(169):
    L$=CHR$(179)+CHR$(153):
    R$=CHR$(166)+CHR$(179):
    F$=CHR$(174)+CHR$(157):
    E$="  "
120 T=0: P=926: X=30: P$=F$: SC=0: LA=15360+544: LP=544: LS=2
130 CLS: FOR A=0 TO 14: PRINT D$(A): NEXT: PRINT D$(A) HS;
140 M=M+1: IF M=1 THEN PRINT @ P,F$; ELSE
    IF M>3 THEN PRINT @ P,E$;: M=0
150 GOTO 220
160 I=PEEK(14400): IF I=0 THEN 140ELSE
    IF I=1 THEN GOTO 270ELSE T=P
170 IFIAND8THENP$=U$:P1=PEEK(15296+P):P2=PEEK(15297+P):
    IFP1<191ANDP2<191THENP=P-64:IFP1=420RP2=42THENSC=SC+1
180 IFIAND16THENP$=D$:P1=PEEK(15424+P):P2=PEEK(15425+P):
```

```
    IFP1<191ANDP2<191THENP=P+64:IFP1=42ORP2=42THENSC=SC+1
190 IFIAND32THENP$=L$:P1=PEEK(15359+P):
    IFP1<191THENX=X-1:P=P-1:IFP1=42THENSC=SC+1
200 IFIAND64THENP$=R$:P1=PEEK(15362+P):
    IFP1<191THENX=X+1:P=P+1:IFP1=42THENSC=SC+1
210 PRINT @ 965,SC;: PRINT @ T,E$;:
    IF TG THEN TG=0: PRINT @ P,F$;: ELSE TG=1: PRINT @ P,P$;
220 POKE LA,32: L3=PEEK(LA+LS):
    IF L3<191 THEN IF ABS(LP-P)<2 THEN GOTO 260 ELSE
    LP=LP+LS: LA=LA+LS: POKE LA,43: GOTO 160
230 IF ABS(LS)<3 THEN IF LP>PTHEN LS=-64 ELSE
    LS=64 ELSE IF LP-INT(LP/64)*64>X THEN LS=-2 ELSE LS=2
240 IF PEEK(LA+LS)=191 THEN LS=-LS
250 GOTO 220
260 FOR A=1 TO 15: PRINT @ P-2, "GOTCHA";: FOR B=1 TO 60: NEXT:
    PRINT @ P-2, "      ";: FOR B=1 TO 60: NEXT: NEXT
270 IF SC>HS THEN HS=SC
280 PRINT @ P-2, "AGAIN";: INPUT Z$: GOTO 120
```

GRAPHICS

Dynamic Graphics with Pool Ball
Recreating Graphics
Super Fast Graphics in BASIC

Dynamic Graphics with Pool Ball

by David L. Kahn

When I demonstrate my TRS-80 to my friends, there are few things that are more impressive than graphics. Unfortunately, many graphics programs require that the user do a lot of work, either playing a game or specifying parameters. To fill what I saw as a void, I wrote Pool Ball, a short (24 lines, 662 bytes) program that produces interesting graphics, allowing owner and guests to sit back and watch.

Pool Ball is a dynamic simulation of a pool ball rolling around a pool table. Watching it run is more interesting than seeing the result. As the ball rolls, it leaves a white track behind itself, producing intricate patterns. By varying the six input parameters, you can produce different patterns. Figure 1 shows a sample screen. All screen pictures are black on white, instead of the white on black that appears on the screen.

Program Design

I designed Pool Ball to be simple and short. No user interaction is necessary after you specify the input parameters. It is dynamic and reasonably fast.

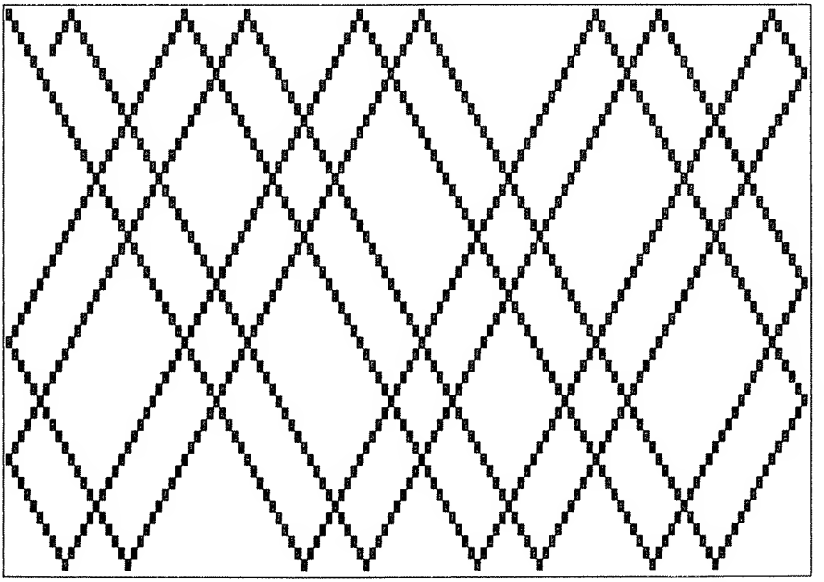


Figure 1. *A sample pattern produced by the Pool Ball program, produced using the default values of all parameters. This snapshot was taken about 40 seconds after the start.*

To meet these objectives, I chose a number of simplifying design alternatives. The rolling ball would leave a trail of individual graphics dots. The ball would always travel at a 45 degree angle to the edges, although this is horizontally compressed on the screen. The user view of the pool table is from directly above.

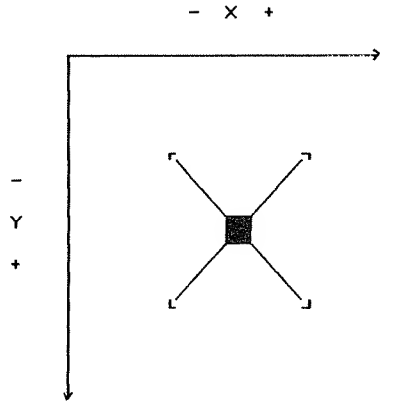


Figure 2. *The four possible ball directions. The ball is moved by adding either +1 or −1 to both the x- and y-coordinates. A +1 produces motion to the right (x) or down (y). A −1 moves to the left (x) or up (y).*

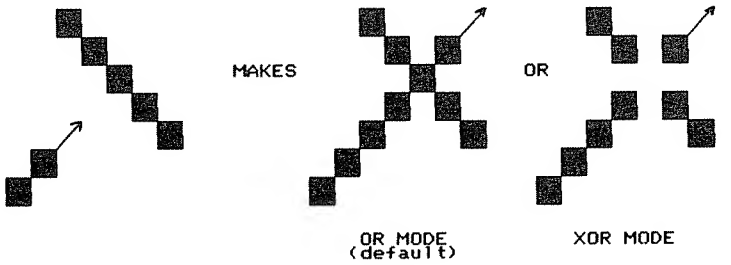


Figure 3. *The two graphics modes. Normally the program runs in the OR mode, drawing a path behind itself. When the XOR (exclusive OR) mode is selected, the ball complements whatever pattern it crosses.*

The design I chose requires eight parameters. The dimensions of the table and the position of the ball are described using Cartesian (x,y) coordinates, where x increases to the right, and y increases downwards on the screen. Figure 2 describes the coordinate system and the four possible ball directions. As the ball moves, it leaves its trail in one of two graphics modes, as shown in Figure 3. The OR mode always leaves a white trail, while the XOR (exclusive OR) mode complements the pattern as it crosses. When the ball reaches an edge, it can do one of two things, as indicated by the edge mode. The two edge modes, Non-Stick and Stick, are illustrated in Figure 4. Two

parameters are required to indicate the current direction of the ball, as a Cartesian vector (x,y). By setting the x and y components of this vector equal to either +1 or −1, adding the vector to the current position produces motion along a 45 degree angle.

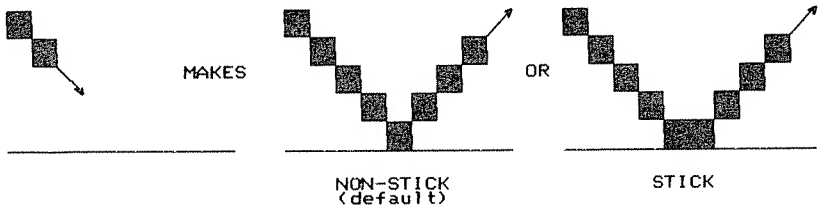


Figure 4. *The two edge modes. Normally the program runs with an immediate edge reflection. When the Stick mode is selected, the ball sticks at the edge for one cell.*

The Program

When you run the program, it asks for the values of each of the first six parameters. The direction vector is initialized to (+1, +1). Fixing the initial direction causes no lack in flexibility. Any image can still be produced, because of the symmetry of the table. After you answer the questions, Pool Ball draws the pictures.

Refer to Figures 5 and 6 for the description of variables and the statement map. There is one variable for each of the eight parameters. The placement of the input subroutine after the main loop was largely the result of evolution. The program originally had fixed parameters. I replaced the parameter initialization in line 10 with the GOSUB that is now there and tacked the input statements onto the end.

Variable	Possible Values	Name	Description
DX	either +1 or −1	delta X	added to X on every iteration
DY	either +1 or −1	delta Y	added to Y on every iteration
X	0 <= X <= MX	X	current x-coordinate of pool ball
Y	0 <= Y <= MY	Y	current y-coordinate of pool ball
MX	0 <= MX <= 127	max X	number of columns −1
MY	0 <= MY <= 127	max Y	number of rows −1
MO	0 or 1 (non-zero)	mode	0 = OR, 1 (non-zero) = XOR
ST	0 or 1 (non-zero)	stick	0 = don't stick, 1 (non-zero) = stick

Figure 5. *Variables used in the Pool Ball program*

I used several tricks of Level II BASIC in writing Pool Ball. The first is the input default specification in lines 100, 130, 160, 180, and 190. I set the variable to the default value before executing the input statement. If you

press ENTER without entering a number, the default value is retained. The syntax allows the default and input to fit easily on one line.

In lines 40, 50, 70, and 80 I use nested IF statements. The second IF is part of the ELSE clause from the first IF. This allows the handling of each dimension (x,y) and each direction (+ 1, − 1) in only one line.

Lines	Description
5–10	Initialization
20–90	Main loop
20	Leave track
30–50	Adjust column
60–80	Adjust row
90	Go to start of loop
100–200	Input subroutines
100–120	Width of table (rows)
130–150	Length of table (columns)
160–170	Initial ball position
180	Graphics mode
190	Stickiness
200	Return to program

Figure 6. *Statement map of the Pool Ball program*

The Results

Run the program and press ENTER in answer to each of the five questions. You see a track of white start at the top left corner and bounce off the bottom edge, and then the top edge, then the right edge, and so on. Press SHIFT@ at any time to freeze the image; press any key to continue. The pattern changes for several minutes, leaving a checkerboard pattern. This is the default pattern. The design in Figure 1 occurs about 40 seconds into the run.

By varying the input parameters, you can generate many interesting patterns. While most of the following examples use relatively small table sizes, I encourage you to experiment with all sizes.

Example 1

Figure 7 shows several pictures of a simple run using the default modes on a smaller table than Figure 1. This table is 12 by 32, or 1/16 of the largest (default) size of 48 by 128. Where ENTER appears in these figures, merely press the ENTER key.

The input parameters are described at the top of Figure 7. Immediately below that, a picture of the screen is shown just after the start. After passing through the next two stages as shown, the program reaches the screen

shown at the bottom of the figure. Once this pattern is reached, it no longer changes.

```
HOW MANY ROWS? 12
HOW MANY COLUMNS? 32
INITIAL POSITION? <ENTER>
ENTER 1 FOR XOR? <ENTER>
ENTER 1 TO STICK AT EDGE? <ENTER>
```

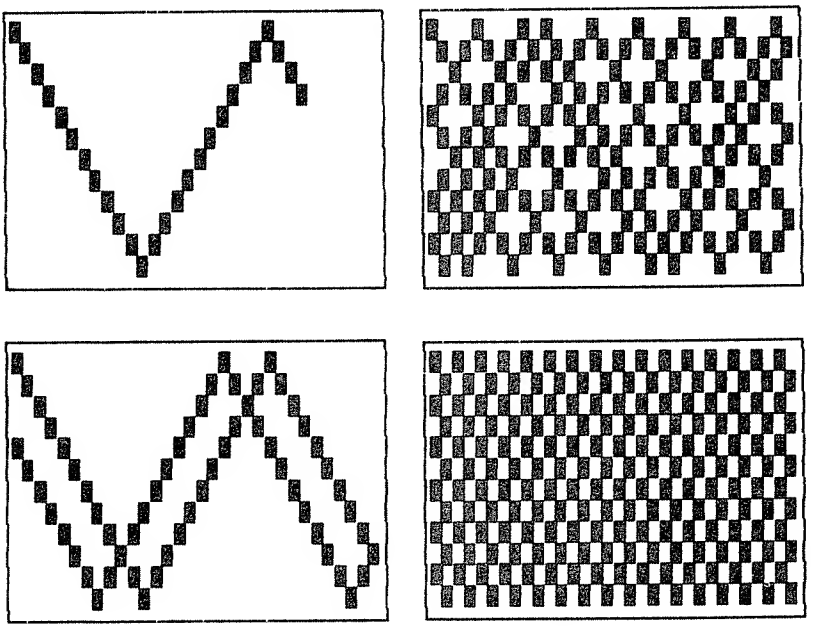
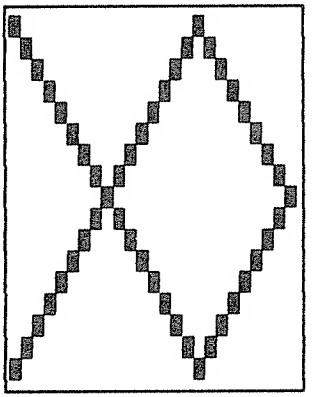


Figure 7. *Several snapshots of a run using default graphics and edge modes*

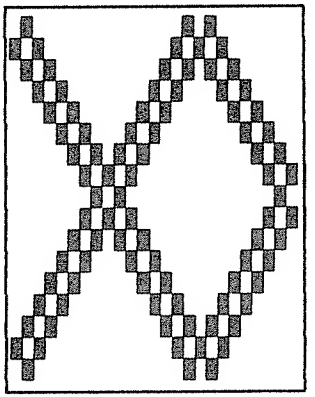
For any table size and starting position (the first three questions), if the defaults are chosen for the last two questions, the final pattern is stable and never becomes more solid than a checkerboard. This is because the ball always rolls as a bishop in chess moves, never leaving its own color. If you imagine a large chess board with a bishop on it, moving diagonally and bouncing off the edges, you will get the picture.

Note that not all sizes produce a solid checkerboard. Figure 8 shows the final result for a 17 by 25 table using three different starting positions and the default modes. Though the checkerboard is not full, once these patterns are reached, the changes stop. Try other positions on the top row as starting positions. Note the transitions from (0,1) to (0,2) and so on to (0,4). As (0,8) is approached, the pattern approaches the original image but is reversed (left to right). What happens when starting positions in other rows are used? (1,1) and (2,2) all produce the same image as (0,0). That is because they are

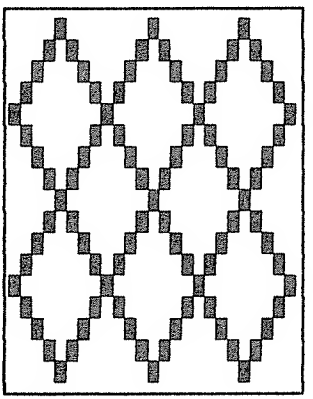
on the same pattern. Any of the patterns in Figure 8 can be produced, starting at any position on the pattern.



HOW MANY ROWS? 17
HOW MANY COLUMNS? 25
INITIAL POSITION? <ENTER>
ENTER 1 FOR XOR? <ENTER>
ENTER 1 TO STICK AT EDGE? <ENTER>



HOW MANY ROWS? 17
HOW MANY COLUMNS? 25
INITIAL POSITION? 0 , 1
ENTER 1 FOR XOR? <ENTER>
ENTER 1 TO STICK AT EDGE? <ENTER>



HOW MANY ROWS? 17
HOW MANY COLUMNS? 25
INITIAL POSITION? 0 , 4
ENTER 1 FOR XOR? <ENTER>
ENTER 1 TO STICK AT EDGE? <ENTER>

Figure 8. *Final patterns using default modes from different initial positions*

These two properties always hold where the ENTER key is pressed in response to the last two questions (XOR and STICK). The ball never leaves its color on the imaginary board, and any pattern can be produced from any point on it. This can be proven mathematically, but the proof is beyond the scope of this article. These patterns always become stable.

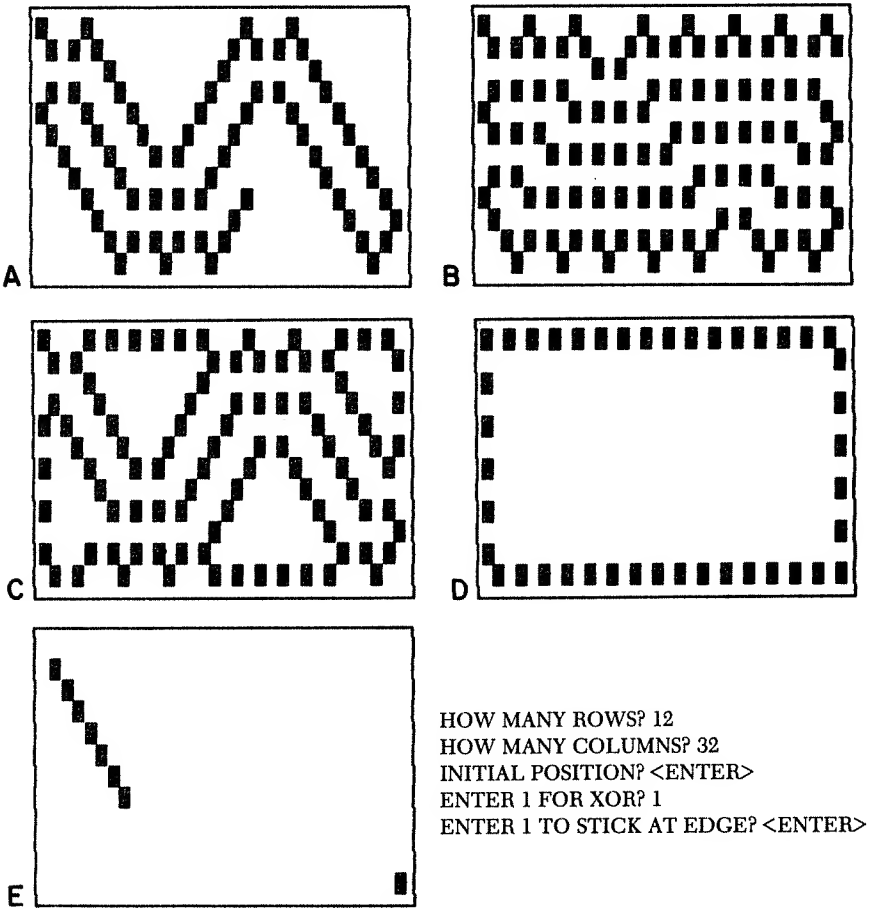


Figure 9. *Several snapshots of a run using XOR graphics mode*

Example 2

What happens when XOR mode is used? Figure 9 shows several snapshots of the patterns produced using the same parameters as in Figure 7, except that the XOR mode is used. The pattern starts off the same, but as soon as the ball crosses its own path, it erases the intersection. Very soon, pattern A in Figure 9 is produced. Patterns B, C, and D occur in sequence during the

first 18 seconds. The border shown in pattern D is produced because each of the internal cells is crossed twice, once along each diagonal, while each of the border cells is touched only once. You can see this while watching the pattern evolve.

After another 18 seconds, most of the screen has been erased, and the pattern starts again, as in pattern E of Figure 9. The only difference is the block in the bottom right corner. When a ball bounces into a corner and back out, it touches most of the cells twice (once each way), but touches the corner only once. Allow the pattern to run for another 36 seconds, and the pattern will start again at the beginning, without the stray block.

Because the ball continues to move like a bishop, it stays on its own color. Since each cell is toggled on at each passing, the pattern never stabilizes. Instead, every pattern repeats exactly after some period of time. The edge and corner properties typically cause a number of near repeats to occur also.

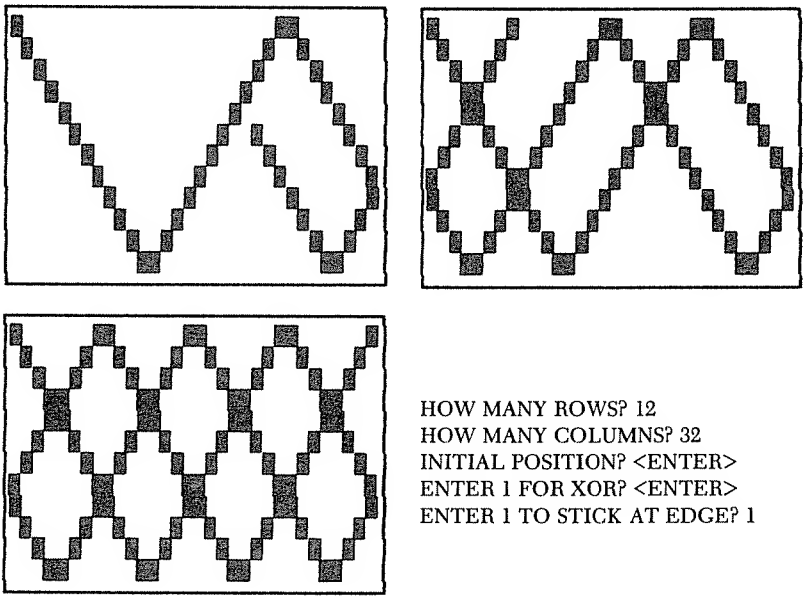
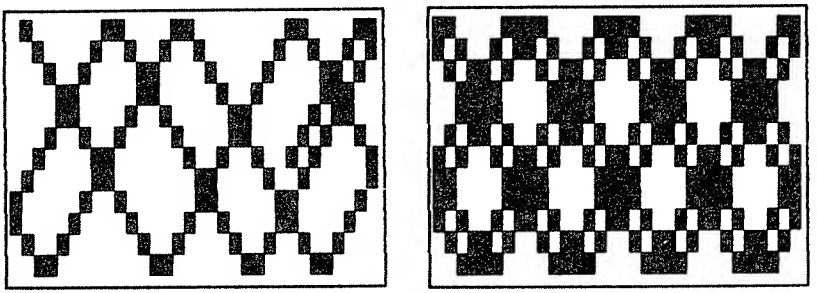


Figure 10. *Several snapshots of a run using the Stick edge mode*

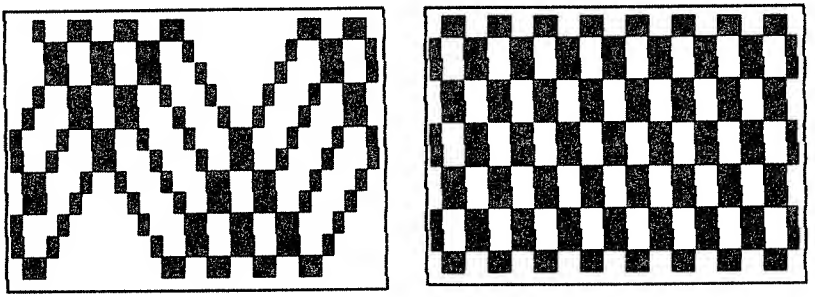
Example 3

What happens when the Stick mode is used? Figure 10 shows three snapshots of the pattern produced using the same table size as in Figures 7 and 9, but with the Stick mode. After passing through the first two images, the pattern stabilizes on the final image. There are several interesting characteristics of this mode. Note that the ball continues to move like a

bishop, except that it changes color every time it hits an edge. The color remains unchanged when the ball hits a corner. The ball always crosses its own path using the opposite color, because the ball bounces off the edge an odd number of times. Can you see why this happens? Compare the bottom snapshot of Figure 10 with the bottom snapshot of Figure 8.



```
HOW MANY ROWS? 12
HOW MANY COLUMNS? 32
INITIAL POSITION? 0 , 1
ENTER 1 FOR XOR? <ENTER>
ENTER 1 TO STICK AT EDGE? 1
```



```
HOW MANY ROWS? 12
HOW MANY COLUMNS? 32
INITIAL POSITION? 0 , 2
ENTER 1 FOR XOR? <ENTER>
ENTER 1 TO STICK AT EDGE? 1
```

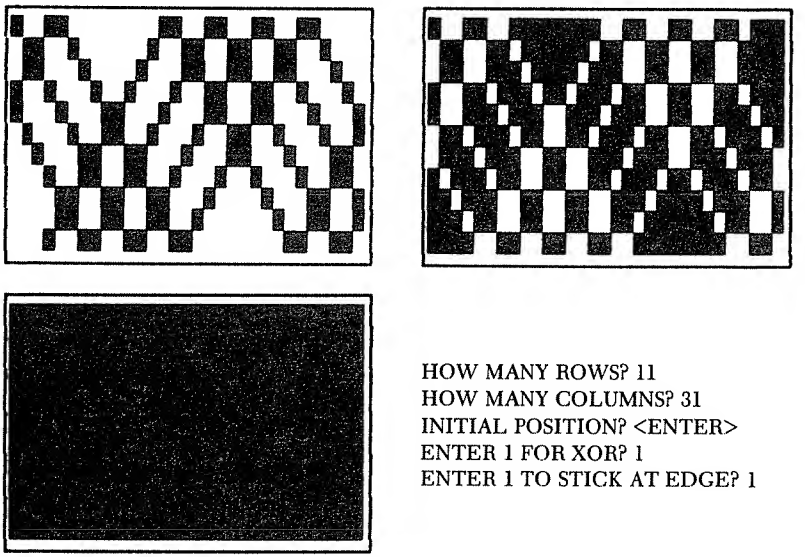
Figure 11. *Intermediate and final patterns using the Stick mode from different initial positions*

When you change the starting position from the default to (0,1) and (0,2), the intermediate and final results are shown in Figure 11. Unlike Figure 8 of Example 1, these patterns are not as obviously related, and even share common points. The patterns always become stable, since no cells are ever turned off.

Example 4

By selecting both the XOR and Stick modes, you can create more patterns. The patterns oscillate, because the image is complemented, and cover both colors of the imaginary checkerboard. It may or may not fill the entire board, depending on the board size and starting position. If a 12 by 32 board is used, the results are similar to Figure 10 of Example 3. When the ball hits the upper right-hand corner, it sticks, then bounces back and starts erasing itself, all the way back to the top snapshot, and then starts again.

By reducing the size of the table to 11 by 31, you produce the results shown in Figure 12. After the image of the first snapshot and the second, the entire board fills up. Then it starts erasing itself, and returns to the initial image. This pattern continues to oscillate. Notice that the top two images are not quite color complements of each other.



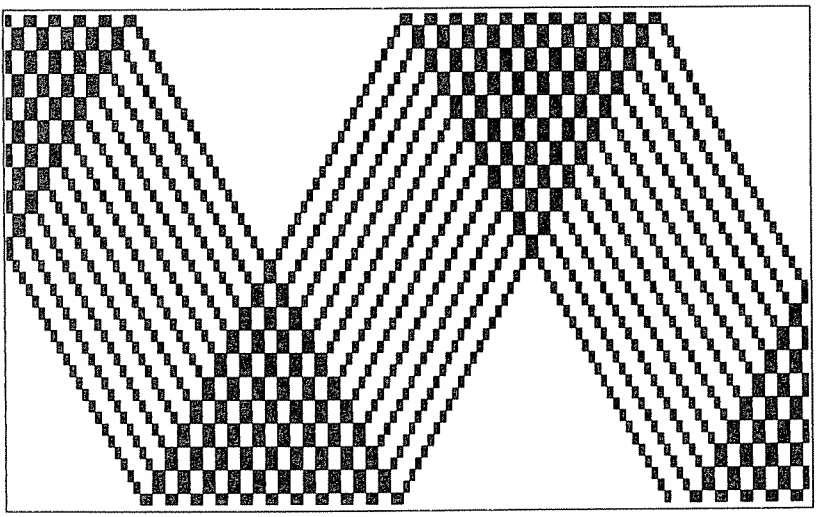
```
HOW MANY ROWS? 11
HOW MANY COLUMNS? 31
INITIAL POSITION? <ENTER>
ENTER 1 FOR XOR? 1
ENTER 1 TO STICK AT EDGE? 1
```

Figure 12. *Several snapshots of a run using both the XOR and STICK modes*

As the preceding examples show, it is easy to generate a variety of interesting graphics patterns using a very simple program. As you experiment, many of the more predictable characteristics of the patterns become obvious. As the examples illustrate, the XOR mode causes the patterns to oscillate, while the Stick mode causes a color shift along the edges. Varying the starting position of an uninteresting pattern often produces a very interesting one.

I recommend experimenting with larger tables such as those in Figures 1 and 13. The richness of the variations on boards of these sizes is much greater.

For those of you with printers, it is a simple matter to insert a trap in the program (mine is between lines 20 and 30) to a subroutine that scans the screen and prints it. Those who have TRS-80 or dot graphics can use them, while asterisks work well for those who don't.



```
HOW MANY ROWS? 42
HOW MANY COLUMNS? <ENTER>
INITIAL POSITION? <ENTER>
ENTER 1 FOR XOR? 1
ENTER 1 TO STICK AT EDGE? 1
```

Figure 13. *Sample image using both the XOR and STICK modes on a large table*

Program Listing. *Pool Ball*

```
2 REM  **** POOL BALL ****
5 DEFINT A-Z
10 CLS:GOSUB 100:CLS:DX=1:DY=1
20 IF MO AND POINT(X,Y) RESET(X,Y) ELSE SET(X,Y)
30 IF DX = 1 THEN 50
40 IF X > 0 THEN X=X-1 ELSE DX=1: IF NOT ST THEN X=1
45 GOTO 60
50 IF X < MX THEN X=X+1 ELSE DX=-1: IF NOT ST THEN X=MX-1
60 IF DY = 1 THEN 80
70 IF Y > 0 THEN Y=Y-1 ELSE DY=1: IF NOT ST THEN Y=1
75 GOTO 90
80 IF Y < MY THEN Y=Y+1 ELSE DY=-1: IF NOT ST THEN Y=MY-1
90 GOTO 20
100 MY=48:INPUT "HOW MANY ROWS (1-48)";MY
110 IF MY < 1 OR MY > 48 GOTO 100
120 MY = MY-1
130 MX=128:INPUT "HOW MANY COLUMNS (1-128)";MX
140 IF MX < 1 OR MX > 128 GOTO 130
150 MX = MX-1
160 X=0:Y=0:INPUT "INITIAL POSITION (ROW,COL)";Y,X
170 IF Y < 0 OR Y > MY OR X < 0 OR X > MX GOTO 160
180 MO=0: INPUT "ENTER 1 FOR XOR";MO
190 ST=0: INPUT "1 TO STICK AT EDGE";ST:IF ST <> 0 THEN ST=-1
200 RETURN
```

Recreating Graphics

by Steve Carr

The idea of writing a TRS-80 program whose output was a BASIC program intrigued me. I wanted this computer-produced program to run correctly and also have practical applications.

I settled on the idea of having the computer speed up my graphics and even write graphics programs from scratch. Using the SET(X,Y) statement can be a painfully slow way to produce a picture. A much faster method is concatenating the picture into a string array by using CHR$ and the ASCII codes for graphics. You can then arrange these strings properly on the screen by using the PRINT@ command. Drawing your picture one block at a time and then looking up the ASCII code for each block is tedious work. Since the computer can do such work quickly, the job seemed perfect for a TRS-80.

Program Listing 1 includes a routine in lines 80–440 to help you draw your picture. You begin with a clear screen, then construct your drawing by maneuvering a flashing pixel about the screen. You direct the flashing light by pressing the four arrow keys. To move the pixel without leaving a line of lighted blocks, hold down the space bar while you press the arrow.

To add characters from the keyboard to your drawing, press the D key. This transfers control to line 310. Since this transfer is brought about by an INKEY$ (line 100), you may have to press the D key more than once. When the transfer to line 310 has been made, a flashing block appears in the upper left-hand corner of the screen. This block is also controlled by the arrows. To insert a character on the screen, hit the appropriate key. You can move the flashing block over previously lit areas without changing them. To return to line 100 (drawing the graphics), press the CLEAR key. No, this does not erase your drawing. You must engage this graphics part of the routine before proceeding in order to save your drawing.

When the drawing is complete, press the A key. This transfers control to line 20030. Once again, you may have to press the key several times. Now the graphics recreating process is at work, scanning the screen and preparing to print out the completed program which, when typed in, reproduces your drawing quickly and accurately.

The drawing in Figure 1 is an example of what this program does. After the drawing was completed on the screen, the graphics recreating process did its work and printed out the program in Program Listing 2.

To convert a drawing created by SET and RESET to one that is produced by string graphics, it is necessary to merge the two programs—program 1

which produces the graphics, and program 2, lines 20000–20850 in Program Listing 1.

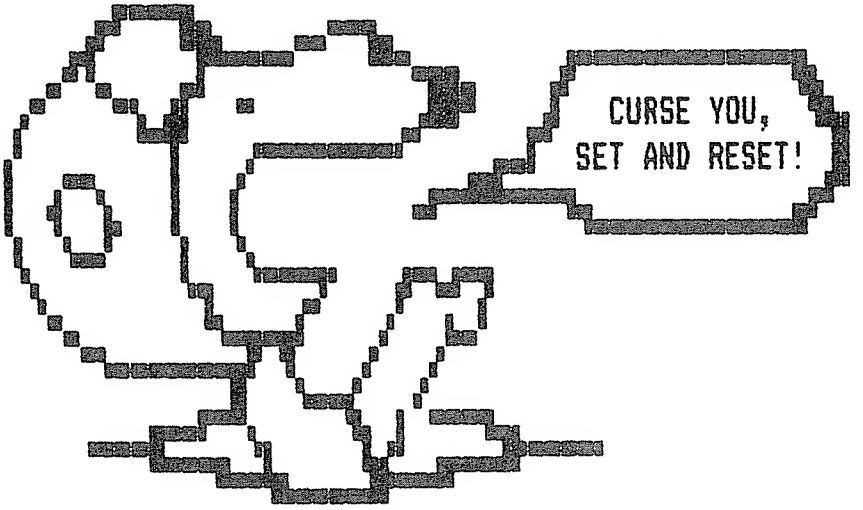


Figure 1. *Picture created by Program Listing 2*

The portion of the program which recreates the graphics is in three sections. The first section, contained in lines 20020–20210, scans the video one line at a time and determines if each line has at least one block lit. If so, the number of characters to be printed, and the position of the first character are both recorded. Lines 20660–20850 contain the program to be printed on paper. Lines 20260–20650 determine and print out the actual data bits that compose your drawing.

Section 1

The video scan is composed of two nested loops. On the TRS-80, the video memory begins at memory address 15360, and each screen line consists of 64 bytes. Thus, in line 20050, the index X is the position in memory of the first block on the line. This loop moves the screen scan vertically. The horizontal movement is provided by the index Z in the inside loop (line 20080). By PEEKing at memory location X + Z, you can examine every block on the screen, one at a time. Table 1 lists the program variables.

Now that the scan is working, you must determine what you are looking at, and if you should do something about it. What the scan finds is simply the ASCII representation of whatever graphics character occupies the current block on the screen. A blank block is represented by ASCII code 32. If these blanks occur at the beginning of a line, they are counted, otherwise

they are ignored. The variable FIRST starts off equal to X (the beginning of the line). For each leading blank found, FIRST increases by 1. When the first non-blank character is found, FIRST becomes the PRINT@ position for the characters on that line. These PRINT@ positions are stored in array PA(COUNT). The length of the line (the number of characters between the first non-blank and the last non-blank character) is stored in the array LNGTH(COUNT).

Section 1: Lines 50–290

P	Determined by PEEKing at 14590. This tells which arrow keys are pressed.
C	The number contained by P, only in a reduced form.
X,Y	Graphic coordinates.
Z	ASCII code for screen block that is replaced by flashing cursor.
B	ASCII code for INKEYed character.
GP	Graphics position; that is, current location of flashing cursor.

Section 2: Lines 20000–20610

COUNT	Number of lines on the screen that have a block lit up.
LN	The line number used in the printout.
X	Control for horizontal screen scan.
Z	Control for vertical screen scan.
FLAG	Set to 0 when first non-blank character is found on a line.
LAST	Position of last non-blank character on a line.
FIRST	Position of first non-blank character on a line.
LNGTH( )	Number of non-blank characters on each line.
PA( )	PRINT@ position for each line.
LIN	Index to keep track of screen line.
NUM	Counter keeping track of how many data points have been printed.
KOUNT	Number of DATA points per line of output.
LO	ASCII code for scanned block.
DUP	Number of times a character has been duplicated.

Table 1. *Program variables*

Section 2

NUM is converted to a character, and the characters for each line are concatenated into the array A$(X) (line 110 of the output). If NUM is negative (the signal for a repeated character), then the absolute value of the next ASCII number (stored in CHAR) is converted to characters and concatenated. When NUM equals 999, the computer begins concatenating the next line. All that remains then is the printing of the strings at their respective PRINT@ positions.

Section 3

The second scan determines and prints out the ASCII code for your graphics. To shorten the output, the program scans ahead to see if a certain character is repeated several times. If so, the number of times it appears is printed out as a negative number, followed by the actual ASCII number. When the scan reaches the end of a line, the flag 999 is printed, and the computer begins again on the next line. For easier readability, only 13 data points are printed on each line of output. The subroutines in lines 20550–20650 convert each data point to a character string and lengthen or shorten the string so that each string is three bytes long. This causes the numbers to be printed in columns, which makes for easier reading.

Program Listing 1. *Graphics recreating program*

```
10 CLEAR 2000
20 DIM PA(16),LNGTH(16)
30 CLS:GP=15360
40 PRINT:PRINT TAB(23)"GRAPHICS RECREATOR":PRINT:PRINT
50 PRINT"DO YOU NEED INSTRUCTIONS?";
60 Z$=INKEY$:IF Z$="" THEN 60
70 IF Z$="Y" THEN PRINT" YES":FOR Z=1 TO 50:NEXT:GOSUB 21000:ELSEPRINT
     " NO":        FOR Z=1 TO 50:NEXT
80 'BEGIN SKETCH
90 CLS
100 IF INKEY$="D" THEN 310 ELSE IF INKEY$="A" THEN 20030
110 P=PEEK(14590):IF P<3 OR P=24 OR P=96 THEN 100 ELSE IF P=128 THEN S
     ET(X,Y):GOTO 190
120 ' DETERMINE WHICH KEYS ARE BEING PRESSED
130 IF P>128 THEN C=INT(SQR(P-127)) ELSE C=INT(SQR(P+1))
140 IF C<3 THEN 100
150 IF P=72 OR P=200 THEN GOSUB 280 ELSE ON C-2 GOSUB 220,230,240,250,
     260,270,290
160 IF Y<0 THEN Y=0 ELSE IF Y>47 THEN Y=47
170 IF X<0 THEN X=0 ELSE IF X>127 THEN X=127
180 'IF SPACE BAR ISN'T PRESSED, LIGHT UP BLOCK; ELSE FLASH IT ON AND
     OFF.
190 IF P<81 THEN SET(X,Y) ELSE SET(X,Y):FOR J=1 TO 7:NEXT:RESET(X,Y)
200 GOTO 100
210 ' CHANGE 'X' AND 'Y' COORDINATES
220 Y=Y-1:RETURN
230 Y=Y+1:RETURN
240 X=X-1:RETURN
250 Y=Y-1:X=X-1:RETURN
260 Y=Y+1:X=X-1:RETURN
270 X=X+1:RETURN
280 Y=Y-1:X=X+1:RETURN
290 Y=Y+1:X=X+1:RETURN
300 ' ADD CHARACTERS TO DRAWING
310 Z=PEEK(GP):POKE GP,143:POKE GP,Z
320     IN$=INKEY$:IF IN$="" THEN 310
330     B=ASC(IN$)
340 ' CLEAR KEY PRESSED?
350     IF B=31 THEN 100
360 ' PRINT INPUTTED CHARACTER ON SCREEN
370 IF B>31 AND B<91 THEN POKE GP,B:GP=GP+1:GOTO 430
380 ' MOVE CURSOR
390     IF B=9 THEN GP=GP+1:GOTO 430
400     IF B=8 THEN GP=GP-1:GOTO 430
410     IF B=91 THEN IF GP-64>15360 THEN GP=GP-64
420     IF B=10 THEN IF GP+64<16383 THEN GP=GP+64
430     IF GP<15360 THEN GP=15360 ELSE IF GP>16383 THEN GP=16383
440 GOTO 310
20000 '
20010 '
20020 ' BEGIN GRAPHIC RECREATOR
20030 COUNT=0:LN=1000
20040 ' X CONTROLS HORIZONTAL MOVEMENT OF THE SCAN
20050 FOR X=15360 TO 16320 STEP 64
20060   FLAG=1:LAST=X:FIRST=X
20070 ' Z CONTROLS VERTICAL MOVEMENT OF THE SCAN
20080   FOR Z = 0 TO 63
20090 ' IF SCANNED CHARACTER IS NOT A LEADING BLANK, GOTO 20090
20100     IF ((PEEK(X+Z)>32) AND (PEEK(X+Z)<>128) OR (FLAG=0)) GOTO 20
     130
20110       FIRST=FIRST+1
20120       GOTO 20150
20130     FLAG=0
20140     IF PEEK(X+Z) <> 32 AND PEEK(X+Z)<>128 THEN LAST=X+Z
20150   NEXT Z
20160 ' IF LINE IS BLANK, SKIP TO 20160
```

Program continued

```
20170     IF LAST=X GOTO 20210
20180        COUNT=COUNT+1
20190        LNGTH(COUNT)=LAST-FIRST+1
20200        PA(COUNT)=FIRST
20210 NEXT X
20220 ' PRINT ON PAPER THE HEART OF COMPUTER GENERATED PROGRAM
20230 GOSUB 20670
20240 '
20250 '
20260 ' BEGIN TO PRINT ON PAPER THE ASCII DATA
20270 FOR LIN = 1 TO COUNT
20280   NUM=0:KOUNT=0
20290 ' PRINT OUT THE 'PRINT @' POSITION
20300   PA=PA(LIN)-15360:GOSUB 20590
20310   LPRINT LN;"  DATA ";PA$;"
          'PRINT @ POSITION"
20320   LN=LN+10
20330   LPRINT LN;"  DATA ";
20340 ' ALL DATA FOR CURRENT VIDEO LINE BEEN PRINTED OUT?
20350   IF NUM = LNGTH(LIN) GOTO 20440
20360     IF KOUNT > 12 THEN LPRINT:LN=LN+10:LPRINT LN;"  DATA ";:KOUN
     T=0
20370     IF KOUNT<>0 THEN LPRINT", ";
20380     LO = PEEK(PA(LIN)+NUM)
20390 ' IS SCANNED BLOCK AND THE ONE BESIDE IT THE SAME?
20400     IF PEEK(PA(LIN)+NUM+1)=LO GOSUB 20470  ELSE GOSUB 20550:LPRI
     NT LO$;
20410     NUM=NUM+1
20420     KOUNT=KOUNT+1
20430   GOTO 20350
20440 LPRINT", 999":LN=LN+10:NEXT LIN
20450 STOP
20460 ' BEGIN ROUTINE TO COUNT UP REPEATED CHARACTERS
20470 DUP=-1
20480 IF PEEK(PA(LIN)+NUM)<>PEEK(PA(LIN)+NUM+1) OR NUM=LNGHT(LIN) THEN
      LO=PEEK(PA(LIN)+NUM):GOSUB 20570: GOSUB 20550:LPRINT DUP$;", ";L
     O$;:KOUNT=KOUNT+1:RETURN
20490 DUP=DUP-1
20500 NUM=NUM+1
20510 GOTO 20480
20520 '
20530 '
20540 '
20550 'CONVERT LO TO LO$
20560 X=LO:GOSUB 20620:LO$=X$:RETURN
20570 'CONVERT DUP TO DUP$
20580 X=DUP:GOSUB 20620:DUP$=X$:RETURN
20590 'CONVERT PA TO PA$
20600 X=PA:GOSUB 20620:PA$=X$:RETURN
20610 '
20620 '  MAKE ALL DATA NUMBERS INTO A THREE CHARACTER STRING
20630 X$=STR$(X):LX=LEN(X$)
20640 IF LX>3 THEN X$=RIGHT$(X$,3) ELSE IF LX<3 THEN X$=" "+X$
20650 RETURN
20660 ' PART OF PROGRAM TO BE PRINTED ON PAPER
20670 LPRINT"  50    CLS"
20680 LPRINT"  60    CLEAR 1000"
20690 LPRINT"  70    DIM PA(16),A$(16)"
20700 LPRINT"  80    FOR X = 1 TO ";COUNT
20710 LPRINT"  90      READ PA(X), NUM"
20720 LPRINT"  100     IF NUM = 999 THEN GOTO 140"
20730 LPRINT"  110       IF NUM < 0 GOSUB 500 ELSE A$(X)=A$(X)+CHR$(NU
     M)"
20740 LPRINT"  120     READ NUM"
20750 LPRINT"  130     GOTO 100"
20760 LPRINT"  140   NEXT X"
20770 LPRINT"  150   FOR X = 1 TO ";COUNT
20780 LPRINT"  160     PRINT @ PA(X),A$(X)"
20790 LPRINT"  170   NEXT X"
20800 LPRINT"  180   END"
```

```
20810 LPRINT"  500   NUM=ABS(NUM)"
20820 LPRINT"  510   READ  CHAR"
20830 LPRINT"  520   A$(X)=A$(X)+STRING$(NUM,CHAR)"
20840 LPRINT"  530   RETURN"
20850 RETURN
21000 CLS:PRINT:PRINT"     TO CONSTRUCT YOUR DRAWING, MANEUVER THE FLA
    SHING PIXEL"
21010 PRINT"ABOUT THE SCREEN BY PRESSING THE ARROWS ON THE KEYBOARD"
21020 PRINT"IN THE DESIRED DIRECTION.  TO MOVE DIAGONALLY, PRESS A"
21030 PRINT"HORIZONTAL ARROW AND THE APPROPRIATE VERTICAL ARROW AT THE
    "
21040 PRINT"SAME TIME.  TO LOCATE YOUR PRESENT POSITION, HOLD THE SPAC
    E"
21050 PRINT"BAR DOWN BRIEFLY.  THIS WILL CAUSE THE PIXEL TO FLASH."
21060 PRINT"TO MOVE THE PIXEL WITHOUT LIGHTING A BLOCK, KEEP THE SPACE
    "
21070 PRINT"BAR DEPRESSED ALONG WITH THE ARROWS."
21080 PRINT:PRINT"     TO ADD CHARACTERS TO YOUR GRAPHICS, PRESS THE '
    D' KEY."
21090 PRINT"YOU MAY HAVE TO PRESS THE 'D' KEY SEVERAL TIMES BEFORE THE
    "
21100 PRINT"COMPUTER TRANSFERS CONTROL TO THE CHARACTER INSERTION ROUT
    INE."
21110 GOSUB 21260
21120 PRINT:PRINT"WHEN THE TRANSFER HAS BEEN MADE, A FLASHING BLOCK WI
    LL APPEAR"
21130 PRINT"IN THE UPPER LEFT-HAND CORNER.  ONCE AGAIN, MOVE THE BLOCK
    "
21140 PRINT"BY PRESSING THE ARROWS.  TO ADD A CHARACTER, SIMPLY HIT"
21150 PRINT"THE DESIRED KEY.  YOU MAY MOVE THE BLOCK OVER PREVIOUSLY"
21160 PRINT"DRAWN GRAPHICS WITHOUT CHANGING THEM.  TO RETURN TO THE"
21170 PRINT"FLASHING PIXEL, PRESS THE 'CLEAR' KEY."
21180 PRINT:PRINT"     TO START THE GRAPHICS RECREATOR, PRESS THE 'A'
    KEY."
21190 PRINT"ONCE AGAIN, IT MAY BE NECESSARY TO PRESS IT SEVERAL TIMES.
    "
21200 PRINT"AT THIS POINT, YOUR PRINTER SHOULD BE ON, BECAUSE THE"
21210 PRINT"COMPUTER WILL SOON BEGIN TO PRINT OUT IT'S PROGRAM."
21220 PRINT" N O T E -- YOU MUST BE IN THE GRAPHICS MODE RATHER THAN"
21230 PRINT"THE CHARACTER INSERTION BEFORE YOU ATTEMPT TO START THE"
21240 PRINT"GRAPHICS RECREATOR.":GOSUB 21260
21250 RETURN
21260 PRINT:PRINT"HIT ANY KEY TO CONTINUE";
21270 Z$=INKEY$:IF Z$="" THEN 21270
21280 CLS:RETURN
```

Program Listing 2

```
50 CLS
60 CLEAR 1000
70 DIM PA(16),A$(16)
80 FOR X=1 TO 11
90 READ PA(X),NUM
100 IF NUM =999 THEN GOTO 140
110 IF NUM < 0 THEN GOSUB 500 ELSE A$(X)=A$(X)+CHR$(NUM)
120 READ NUM
130 GOTO 100
140 NEXT X
150 FOR X=1 TO 11
160 PRINT @ PA(X),A$(X)
170 NEXT X
180 END
500 NUM=ABS(NUM)
510 READ CHAR
520 A$(X)=A$(X)+STRING$(NUM,CHAR)
530 RETURN
1000 DATA 73
```

Program continued

```
1010 DATA  160 , 156 , 135 ,  -2 , 131 , 191 , 188 , 144 ,  -3 ,  32 ,
      128 ,  -2, 176
1020 DATA   -2 , 140 , 172 , 176 , 999
1030 DATA  135
1040 DATA  176 , 142 , 139 , 176 ,  -4 ,  32 , 176 , 151 ,  -5 , 131 ,
       -4 ,  32, 130
1050 DATA   -2 , 131 , 140 ,  -2 , 188 , 176 ,  -3 ,  32 , 128 , 160 ,
      158 ,  -13, 131
1060 DATA  175 , 180 , 999
1070 DATA 197
1080 DATA  152 , 131 ,  -4 ,  32 , 131 , 169 , 176 , 190 , 149 ,  -2 ,
      128 , 130, 129
1090 DATA   -4 , 128 ,  -4 ,  32 , 160 , 152 ,  -2 , 143 , 131 ,  -3 ,
      128 , 176, 158
1100 DATA   -3 ,  32 ,  67 ,  85 ,  82 ,  83 ,  69 ,  32 ,  89 ,  79 ,
       85 ,  44,  -2 ,  32
1110 DATA  139 , 173 , 148 , 999
1120 DATA 260
1130 DATA  168 , 129 ,  32 , 128 ,  -2 , 176 ,  -4 , 128 , 170 ,  -2 ,
      128 ,  32, 160
1140 DATA  134 ,  -8 , 131 , 129 ,  -3 ,  32 , 160 , 176 , 156 , 140 ,
      133 ,  -2,  32
1150 DATA   83 ,  69 ,  84 ,  32 ,  65 ,  78 ,  68 ,  32 ,  82 ,  69 ,
       83 ,  69, 84
1160 DATA   33 , 128 , 186 , 149 , 999
1170 DATA  324
1180 DATA  170 ,  -2 , 128 , 174 ,  -2 , 128 , 169 , 144 ,  -2 ,  32 ,
      170 ,  -3,  32
1190 DATA  149 ,  -8 ,  32 ,  -2 , 128 , 140 , 135 ,  -7 , 131 , 139 ,
      172 , -12, 176
1200 DATA  188 , 135 , 129 , 999
1210 DATA  389
1220 DATA  165 ,  -2 ,  32 , 137 , 140 , 134 ,  -4 ,  32 , 149 ,  -2 ,
      128 , 130, 164
1230 DATA   -4 , 176 , 144 ,  -3 ,  32 , 160 , 176 , 144 , 160 ,  -2 ,
      176 , 999
1240 DATA  454
1250 DATA  137 , 176 ,  32 , 128 ,  -5 ,  32 , 130 , 171 , 144 ,  -4 ,
       32 , 152, 134
1260 DATA   -3 ,  32 , 160 , 135 ,  32 , 130 , 147 ,  32 , 150 , 999
1270 DATA 520
1280 DATA  131 , 140 , 164 ,  -6 , 176 , 178 , 179 , 159 , 131 , 175 ,
      129 ,  -3,  32
1290 DATA  152 , 129 ,  -2 ,  32 , 152 ,  -2 , 131 , 999
1300 DATA 592
1310 DATA   -2 , 176 , 191 ,  -3 , 128 , 137 ,  -2 , 140 , 182 ,  -2 ,
      128 , 144, 134
1320 DATA   -4 , 176 , 999
1330 DATA  649
1340 DATA  136 ,  -3 , 140 , 191 ,  -2 , 179 , 177 , 176 ,  32 , 137 ,
      148 ,  -4,  32
1350 DATA  130 , 181 , 152 , 133 , 128 , 160 ,  -2 , 176 , 178 , 187 ,
      157 ,  -4, 140
1360 DATA  132 , 999
1370 DATA 731
1380 DATA   -3 , 131 , 139 , 141 ,  -4 , 140 , 142 , 135 ,  -3 , 131 ,
      129 , 999
```

Super Fast Graphics in BASIC

by Hardin Brothers

If you have at least one disk drive and any popular disk operating system, you have a pair of powerful screen-handling and graphics commands. Using only BASIC, you have the following abilities:

1) To quickly scroll your screen up, down, left, or right, creating a moving window like that in Scripsit.

2) To create running borders anywhere on the screen that are as fast as those used in machine-language games.

3) To scroll any portion of the screen in any direction, leaving the rest of the screen completely undisturbed.

4) To alternate almost instantly between any two completely different screens.

5) To enter a complete screen of data, text, or graphics into memory from the keyboard without worrying about prompts, the ENTER key, or defining string lengths.

You have probably seen the statements LSET and RSET in the random files section of the DOS manual. This article presents a series of short routines that demonstrate ways of using these commands that the manual does not mention.

BASIC String Handling

The routines involve string manipulation, so a quick review of BASIC's string-handling commands and idiosyncracies is in order. First, enter but do not run the following program:

```
10 ? PEEK(VARPTR(A$) + 1) + PEEK(VARPTR(A$) + 2)*256
20 ? PEEK(VARPTR(B$) + 1) + PEEK(VARPTR(B$) + 2)*256
```

Information about each active string is kept in a memory index above the BASIC program. The index includes the string's name and length, and the address of the string. Literal strings (defined in statements like A$ = "LITERAL") are stored in the program lines in which they are defined. Manipulated strings are kept in the cleared string space at the top of available memory.

VARPTR(A$) is the address of the string length; PRINT PEEK(VARPTR (A$)) prints the length of the string. The two bytes immediately following the string length in the index table contain the address of the string in least significant byte / most significant byte form. Using this short program, you

can obtain the addresses of the active strings A$ and B$ by entering GOTO 10. Do not enter RUN, because it would erase the pointers to the variables A$ and B$.

In the following discussion, the addresses of strings are those obtained using a Model I with 48K RAM and NEWDOS + . The values you find with your computer may be different, but their interpretation is the same. Enter from the Command mode:

```
CLEAR 1000:A$ = "TEST #1":B$ = A$:GOTO 10
```

The numbers 65529 and 65522 appear, indicating that the top seven bytes of memory hold A$, and the next seven hold B$. Now enter:

```
A$ = B$:B$ = A$:GOTO 10
```

A$ is stored at 65515, and B$ is stored at 65508. If you use DEBUG to look at memory, you find that the top 28 bytes contain:

```
TEST #1TEST #1TEST #1TEST #1
```

Even though you have only two active string variables and they are identical, BASIC stores the same text four times.

If you experiment with other string functions, such as LEFT$, RIGHT$, and STRING$, you find that more high memory is used each time a string is manipulated. When no room is left in cleared high memory, BASIC goes through its garbage collecting routine, moving active strings up to the top of memory and ignoring but not erasing memory that is no longer used for active string storage. This is why programs that handle a lot of strings stop so often to rearrange memory—the cleared space is used up quickly.

LSET Makes a Difference

Type in CLEAR 1000 again to clear all variables, then enter:

```
A$ = STRING$(64,32):B$ = "//TEST #2//":GOTO 10
```

A$ occupies the top 64 bytes of memory (starting at 65472) and B$ occupies the next 11 bytes (starting at 65461). Enter:

```
LSETA$ = B$:? LEN(A$):GOTO 10
```

A$ has NOT moved. It is still 64 bytes long, and it is still at the very top of memory. PRINT A$ to see what is in the string. //TEST #2// and 53 blank spaces appear. Enter:

```
B$ = A$:GOTO 10
```

B$ has moved down 64 bytes below its previous address (65397), but A$ still has not moved. Try the same experiments using RSET instead of LSET. The results are the same, but now both A$ and B$ have 53 blank spaces followed by //TEST #2//. As long as you do not put A$ on the left side of the equal sign without LSET or RSET, its address does not change.

When the program encounters LSET A$ = B$, B$ is copied into A$, but neither the position nor the length of A$ changes. IF B$ is longer than A$, B$

is truncated on the right as it is placed in A$. If A$ is longer than B$, LSET adds extra spaces on the right to pad out A$, while RSET adds those extra spaces on the left. In no case, however, does the address or length of A$ change.

Pointing Strings at Video Memory

The first key to super graphics handling is remembering that LSET and RSET do not disturb the address of a string. The string stays at a given spot in memory. The second key is even simpler.

The TRS-80 video display is memory-mapped. This means that, from a programming point of view, anything you put in addresses 15360 to 16383 appears on the screen. The seven (eight with a lowercase modification) video RAM chips look and act like any other 1024 bytes of memory as far as the computer is concerned. When you use POKE graphics, all you are doing is POKEing information into specific addresses in the video RAM. Special circuitry translates any information in the video RAM into a screen display.

If you could make BASIC think that the address of a string was in the video screen area between 15360 and 16383, you could put information onto the screen using LSET and RSET and take information from the screen, without using PRINT, POKE, or PEEK statements. Remember how you found the string address? You can change that address by POKEing whatever values you want into the index. You can also arbitrarily set the length of a string by POKEing a length value into the index.

As an example, enter NEW and then run the following short program:

```
10 CLEAR 1000 : A$ = " "
20 POKE VARPTR(A$),64
30 POKE VARPTR(A$) + 1,15872 - INT(15872/256)*256
40 POKE VARPTR(A$) + 2,INT(15872/256)
```

A$ is defined as a 64-byte string at address 15872, which is the middle line on the video screen. Now enter:

```
B$ = "TEST #3": C$ = STRING$(64,191): RSET A$ = B$
```

TEST #3 should appear on the far right side of the ninth line of the screen. Now try:

```
LSET A$ = C$
```

You get a solid bar across the ninth line of your screen.

The LSET and RSET routines are fast, apparently because they are relatively simple for the computer to carry out. The graphics generated with them are also fast, at least for a BASIC program. Another advantage of using LSET and RSET graphics is that there is no unwanted screen scrolling. Because you do not use the PRINT function, the cursor position does not change, and you are able to print to the lower right-hand corner of the screen without forcing an automatic scroll.

The Scrolling Routines

Each of the routines in the listings is meant to give you some suggestions about how you can use LSET and RSET graphics in your programs. Instead of presenting you with a finished program, I am presenting several short routines from which you can pick and choose. The line numbers in the routines are consecutive, and you can run them together as a single demonstration program.

Program Listing 1 is a housekeeping routine that I have named the Master Module. It performs functions that are necessary for each of the other routines and should precede all other routines in your program. The easiest way to use this routine is to enter it and save it to disk, then call it back before you enter any other listings.

Line 10 clears the screen and then clears a large chunk of string space. When you run graphics programs, you should clear as much memory as the program can afford so that you minimize or eliminate any pauses for garbage collection.

Line 12 defines variables, starting with A through F, as strings, so you do not have to enter a dollar sign for each one. It defines variables G through L as integers to be used in loops and counting sequences. Line 14 sets the dimensions of three string arrays you will use. A(X) is a string array pointed to lines on the video screen; B(X) is a line of text created for demonstration purposes; E(X) is a line of text or data taken from the screen.

The loop in lines 16 through 28 points each of the A(X) variables at a line on the video screen. It is almost identical in operation to the short program presented earlier. You may notice, however, that the first member of the array, A(0), is pointed to memory below the video screen. A(0) is not used in these programs; it is there for insurance. In developing these routines, I found that BASIC sometimes changes the address of the first string in the array and points it to the area of memory usually used for data buffers in disk operations. I don't know the reason for this, but defining an extra string, A(0), solves the problem and requires no extra memory.

Lines 30 to 36 define the 30 members of the B(X) array as text for demonstrations. Lines 38 to 42 set up two graphics strings, C1 and C2, which are used in the running borders demonstrations. Line 46 defines a string, C3, which appears as a solid bar across the screen, and line 48 defines a null string, CL, that is used for clearing any line of the video display.

Program Listing 2 is the simplest routine. Its purpose is to display running borders on the top and bottom of the screen. Each time the program loops through lines 114 through 120, C1 moves one space to the left on the top and bottom lines of the screen, and C2 moves one space to the right on the second and fifteenth lines of the screen. Line 124 sends the program back to line 112 to repeat the process unless there is any keyboard input. Though the routine is simple, the effect on the screen is significant.

Program Listing 3 demonstrates two types of vertical scrolling. Lines 212 to 226 scroll six different lines of text down the screen individually. Each time the program passes through the inner loop (lines 214 to 224), the current line of text is placed one line lower, then the previous line is erased by LSETing a null string (CL) into it. The timing loop in line 222 is shortened for each new line of text. As a result, the first line of text scrolls down the screen relatively slowly, while the last line, line 6, scrolls almost too quickly for you to read.

Lines 232 to 246 are almost identical to the previous lines except that the line of text is scrolled up and the timing loop is adjusted to scroll the first line quickly and the last line slowly.

Lines 252 to 262 scroll 30 lines of text down the screen. K is used to count the number of lines used, and when the thirtieth line of text enters the screen, the routine comes to an end. L counts the number of active screen lines. Each time a new text line is introduced, the number of active screen lines increases until the entire screen is used. Finally, the short loop from 254 to 258 does the actual scrolling and determines which text line is to appear on which screen line.

Lines 284 to 294 scroll the same 30 text lines upward. You could replace this routine by moving the cursor to the beginning of the last screen line and then PRINTing each line of text. Using LSET, however, allows 16 full lines of 64 characters to be shown on the screen. The PRINT statement, because of the automatic scroll whenever a character is printed in the lower right-hand corner of the screen, could never display more than 15 64-character lines.

Scrolling Horizontally

The fourth routine, in Program Listing 4, demonstrates horizontal scrolling, which is usually only found in expensive word-processing programs. To move any line across the screen requires two loops: one to move the text line onto the screen, and another to move it off. As with the vertical scrolls, the horizontal scrolls are first demonstrated with single text lines and then with a full screen.

First, four text lines are scrolled across the screen from right to left in lines 312 to 326. The first loop, lines 314 to 318, RSETs increasing portions of the text line into the eighth screen line. Then, in lines 320 to 324, decreasing portions of the text line are LSET into the screen line. Lines 334 to 344 are similar but, by decreasing instead of increasing the counting variable, H, they move the text lines from left to right. Lines 352 to 370 and 376 to 394 are almost identical except that they are used to scroll a complete screen left and then right.

Program Listing 5 demonstrates how you can use vertical and horizontal scrolling together. It also shows how you can scroll part of the screen

without affecting the rest of the screen. In lines 410 to 418, identical strings of graphics blocks are simultaneously scrolled both up and down to cover two-thirds of the screen. If you would like to see the scrolling more clearly, add a short timing loop between lines 414 and 418. Then, the 30 lines of text are scrolled down through the six-line window left in the middle of the screen. The logic used is almost identical to that of Program Listing 3, except that only six lines instead of 16 are being scrolled.

Lines 432 to 436 capture the present screen in array E(X). Line 434 does not change the addresses of the array A(X) because A(I) is on the right side of the equal sign. If it were on the left of the equal sign, we would lose our ability to LSET strings to the screen. E(X) is used to capture the screen and make the rest of Program Listing 5 independent of the actual data that is on the screen.

Lines 438 to 446 scroll the middle window of text off the screen to the right. Then, lines 448 to 458 scroll the two graphics blocks off in opposite directions. In both cases, the logic is identical to that used in Program Listing 4.

Screen Input and Screen Swap

The last two demonstration routines are shown together in Program Listing 6 because the second routine, the screen swap, uses the screen generated by the full screen input routine for half of its data. The screen swap can also be used with any two full screens or partial screens of data.

I will not describe the screen input routine in great detail because most of it has little to do with LSET/RSET graphics. The routine first asks if you wish to input text (alphanumeric) or graphics data. If you choose graphics, each regular keyboard key is capable of inputting two different graphics blocks at the cursor position, depending on whether the SHIFT key is held down at the same time as the letter. Regardless of which input mode you select, the arrow keys, ENTER, and CLEAR all perform screen editor functions by moving the cursor or clearing the current line.

After the cursor has moved off the bottom of the screen and you have finished entering a full screen of text or graphics, the program falls through to line 582. In the next three lines, just as in Program Listing 5, the screen is put into the E(X) array and thereby stored in high memory.

The screen swap routine, lines 610 to 626, is very simple. The first loop, lines 610 to 614, fills the screen with the dummy text that you have used in each of the other routines. The program then looks for any key to be pressed (line 616), and, when a keystroke is found, the text you entered during the screen input routine is displayed (lines 618 to 622). You can continue to switch between the two screens by pressing any key on the keyboard. This simple routine shows the remarkable speed of LSET graphics.

Using the Techniques

Using these graphics tricks, you can make the graphics in your programs a lot more spectacular. Any program, whether it be a game, a business application, or a personal utility program, looks much more professional with a running border around its title screen. You can use the horizontal and the vertical scrolls for marquees and computer animated movies. You can use the screen swaps for complex games with more than one board or for examining each level of three-dimensional Tic Tac Toe.

All of these routines have applications for data input and handling routines in business or home finance applications or in data-base management programs. There is no reason to restrict yourself to strings such as A(X) that are each one line long. Longer strings of up to 255 bytes can be pointed at the video screen so that up to four lines of data can be LSET or RSET to the screen instantly. Short strings can be pointed at the screen to handle formatted information such as mailing addresses or social security numbers.

Program Listing 1. *Master module*

```
1 '**** LSET/RSET GRAPHICS ****
2 '****    MASTER MODULE   ****
3 '                    WRITTEN BY  HARDIN BROTHERS
4 '                                280 NORTH CAMPUS AVE.
5 '                                UPLAND, CA. 91786
6 '
10 CLS : CLEAR 10000
12 DEFSTR A - F : DEFINT G - L
14 DIM A(16), B(30), E(16)
16 FOR I = 0 TO 16
18     A(I) = " "
20     G = 15360 + (I-1)*64
22     POKE VARPTR(A(I)), 64
24     POKE VARPTR(A(I))+1, G-INT(G/256)*256
26     POKE VARPTR(A(I))+2, INT(G/256)
28 NEXT I
30 FOR I = 1 TO 30
32     B(I) = STRING$(I,"-") + "LINE " + STR$(I)
34     B(I) = B(I) + STRING$(64-LEN(B(I)),"-")
36 NEXT I
38 FOR I = 1 TO 17
40     C1 = C1 + CHR$(191) + "   "
42     C2 = C2 + CHR$(140) + "   "
44 NEXT I
46 C3 = STRING$(64,191)
48 CL = ""
```

Program Listing 2. *Running borders*

```
100 '**** LSET/RSET GRAPHICS ****
101 '****  RUNNING BORDERS   ****
102 '
103 'THIS ROUTINE MUST BE PRECEDED BY THE MASTER MODULE
104 '
110 CLS
111 LSETA(8)=STRING$(19,32)+"PRESS ANY KEY TO CONTINUE"
112 FOR I = 1 TO 4
114    LSETA(1) = MID$(C1,I)
116    LSETA(16) = MID$(C1,I)
118    LSETA(2) = MID$(C2,5-I)
120    LSETA(15) = MID$(C2,5-I)
122 NEXT I
124 IF INKEY$="",112
```

Program Listing 3. *Vertical scrolls*

```
200 '**** LSET/RSET GRAPHICS ****
201 '****  VERTICAL SCROLLS  ****
202 '
203 'THIS ROUTINE MUST BE PRECEDED BY THE MASTER MODULE
204 '
205 ' ** SCROLL A SINGLE LINE DOWN THE SCREEN **
210 CLS
212 FOR I = 1 TO 6
214    FOR J = 1 TO 16
216       K = J - 1 : IF K = 0, K = 16
218       LSETA(K) = CL
220       LSETA(J) = B(I)
222          FOR L = 0 TO (6 - I)*20 : NEXT L
224    NEXT J
226 NEXT I
```

```
230 '** SCROLL A SINGLE LINE UP THE SCREEN
232 FOR I = 1 TO 6
234    FOR J = 16 TO 1 STEP -1
236       K = J + 1 : IF K = 17, K = 1
238       LSETA(K) = CL
240       LSETA(J) = B(I+6)
242          FOR L = 0 TO (I-1)*20 : NEXT L
244    NEXT J
246 NEXT I
250 '** SCROLL 30 LINES OF TEXT DOWN THE SCREEN
252 K = 0 : L = 1
254 FOR J = 1 TO L
256    LSET A(J) = B(30 - K + J - 1)
258 NEXT J
260 L = L + 1: IF L>16 , L=16
262 K = K + 1: IF K<30 THEN 254
280 '** SCROLL 30 LINES OF TEXT UP THE SCREEN
282 CLS
284 K = 0 : L = 16
286 FOR J = 16 TO L STEP -1
288    LSET A(J) = B(K + J -15)
290 NEXT J
292 L = L - 1 : IF L<1, L=1
294 K = K + 1: IF K<30 THEN 286
```

Program Listing 4. *Horizontal scrolls*

```
300 '**** LSET/RSET GRAPHICS ****
301 '**** HORIZONTAL SCROLLS ****
302 '
303 'THIS ROUTINE MUST BE PRECEDED BY THE MASTER MODULE
304 '
305 '** SCROLL A SINGLE LINE LEFT
310 CLS
312 FOR G = 1 TO 4
314    FOR H = 1 TO 63
316       RSET A(8) = LEFT$(B(G),H)
318    NEXT H
320    FOR H = 1 TO 64
322       LSET A(8) = MID$(B(G),H)
324    NEXT H
326 NEXT G
330 '** SCROLL A SINGLE LINE RIGHT
332 FOR G = 5 TO 8
334    FOR H = 64 TO 1 STEP -1
336       LSET A(8) = MID$(B(G),H)
338    NEXT H
340    FOR H = 63 TO 0 STEP -1
342       RSET A(8) = LEFT$(B(G),H)
344    NEXT H
346 NEXT G
350 '** SCROLL A FULL SCREEN LEFT
352 FOR G = 1 TO 63
354    FOR H = 1 TO 16
356       RSET A(H) = LEFT$(B(H),G)
358    NEXT H
360 NEXT G
362 FOR G = 1 TO 65
364    FOR H = 1 TO 16
366       LSET A(H) = MID$(B(H),G)
368    NEXT H
370 NEXT G
374 '** SCROLL A FULL SCREEN RIGHT
376 FOR G = 64 TO 1 STEP -1
378    FOR H = 1 TO 16
380       LSET A(H) = MID$(B(H),G)
382    NEXT H
```

Program continued

```
384 NEXT G
386 FOR G = 63 TO 0 STEP -1
388    FOR H = 1 TO 16
390       RSET A(H) = LEFT$(B(H),G)
392    NEXT H
394 NEXT G
```

Program Listing 5. *Mixed screen routine*

```
400 '**** LSET/RSET GRAPHICS ****
401 '**** MIXED SCREEN ROUTINE **
402 '
403 'THIS ROUTINE MUST BE PRECEDED BY THE MASTER MODULE
404 '
405 CLS
410 FOR I=1 TO 5
412    LSET A(I) = C3
414    LSET A(17-I) =C3
418 NEXT I
420 K = 0 : L = 6
422 FOR J = 6 TO L
424    LSET A(J) = B(30 - K + J - 6)
426 NEXT J
428 L = L + 1: IF L>11,L=11
430 K = K + 1: IF K<30 THEN 422
432 FOR I = 1 TO 16
434    E(I) = A(I)
436 NEXT I
438 FOR G = 63 TO 0 STEP -1
440    FOR H = 6 TO 11
442       RSET A(H) = LEFT$(E(H),G)
444    NEXT H
446 NEXT G
448 FOR G = 1 TO 64
450    FOR H = 1 TO 5
452       LSET A(H) = MID$(E(H),G+1)
454       RSET A(17-H) = LEFT$(E(H),64-G)
456    NEXT H
458 NEXT G
```

Program Listing 6. *Full screen input and storage*

```
500 '**** LSET/RSET GRAPHICS ****
501 '**** FULL SCREEN INPUT  ****
502 '****    AND STORAGE     ****
503 ' MUST BE PRECEDED BY THE MASTER MODULE
505 CLS:PRINT@471,"GRAPHICS OR TEXT?"
506 F=INKEY$ : IF F="" THEN 506
507 IF F="G" OR F="g" THEN G=96 ELSE G=0
508 CLS
509 D = CHR$(1)+CHR$(8)+CHR$(9)+CHR$(10)+CHR$(91)+CHR$(13)+CHR$(31)
510 FOR I=15360 TO 16383
512 E=INKEY$ : IF E="", K=PEEK(I): POKE I,132: POKE I,K:GOTO 512
514 IF INSTR(D,E)=0 THEN POKE I, ASC(E)+G : GOTO 580
516 ON INSTR(D,E) GOTO 520, 530, 540, 550, 560, 550, 570
520 CLS : GOTO 510
530 I = I-1 : POKE I,32 : IF I<15360, I=15360
532 GOTO 512
540 I = I+7 : GOTO 580
550 I = INT(I/64)*64 + 63 : GOTO  580
560 J = INT(I/64)*64 - 64 : IF J<15360, J=15360
562 I = J : GOTO 512
570 FOR J = INT(I/64)*64 TO INT(I/64)*64+63
572    POKE J,32
574 NEXT J
```

```
576 I = INT(I/64)*64
578 GOTO 512
580 IF I>16383 THEN 582 ELSE NEXT I
582 FOR I=1 TO 16
584    E(I) = A(I)
586 NEXT I
600 '**** LSET/RSET GRAPHICS ****
601 '**** SCREEN SWAP ROUTINE ***
602 '
603 ' MUST BE PRECEDED BY BOTH THE MASTER MODULE
604 '    AND SCREEN INPUT ROUTINE (LINES 500 - 586)
605 '
610 FOR I = 1 TO 16
612    LSET A(I) = B(I)
614 NEXT
615 LSETA(8)=STRING$(17,32)+"PRESS ANY KEY TO SWAP SCREENS"
616 IF INKEY$="" THEN 616
618 FOR I = 1 TO 16
620    LSET A(I) = E(I)
622 NEXT
624 IF INKEY$="" THEN 624
626 GOTO 610
```

HARDWARE

EPROM Programmer

EPROM Programmer

by Abel J. Tapia

EPROM programmers command a rather high price, so I built a unit that was inexpensive, simple in design, versatile, and yet contained readily available parts. This article describes the design of the hardware and the software necessary to allow the unit to operate. Table 1 lists the programmer parts; Table 2 lists the power supply parts.

The EPROM that I have chosen to support my ongoing projects is the 2716. At the time of this writing, this EPROM was the most cost-effective on the market in that it offered the highest density for the money. This is based on its price divided by the number of programmable bits ($7.00/16384 bits). The cost per bit is less than 0.042 cents. I made a comparison of the cost of the 2716 EPROM against the other types currently available, and the 2716 won hands down. I ruled out the 2708 because it requires three different power supplies to operate, and its cost per bit at this writing was about 0.055 cents ($4.50/8192 bits). The denser 2732 averages about 0.085 cents per bit ($28.00/32768 bits). I anticipate that the 2732 will drop in price and become cost-effective to the experimenter. In anticipation of this decline in price, I have designed the programmer to be capable of programming this device as well.

EPROM and Its Applications

There are three basic types of Read-Only Memory. ROM is Read-Only Memory that has been mask programmed by the manufacturer and cannot be altered by the user. PROM is Programmable Read-Only Memory that can be field programmed by the user. Once it is programmed, however, it cannot be erased or altered. EPROM is an Erasable Programmable Read-Only Memory. All three types of Read-Only Memory have one thing in common—they can be programmed to perform as permanent memory. This is done by programming data at all the memory address locations of the device. Once data has been programmed into a memory location of one of these devices, it will always read that same information whenever power is turned on to the device and that particular location in memory is addressed. Unlike the RAM which loses its contents when power is turned off, the ROM devices always retain it. This article focuses on the EPROM since it can do the job of either the ROM or the PROM. The EPROM, as the name implies, can be erased many times and reprogrammed. To erase, you expose the

device to shortwave ultraviolet light for a period of about 20 minutes. Special erasers for performing this task are available commercially. Having this capability means that, if you make a mistake during programming, you can erase the EPROM and start over. With PROM, there is no second chance; one programming mistake, and the device winds up in the trash.

There are numerous applications for Read-Only Memory devices. One familiar application is to use Read-Only Memory to hold a computer program. In the TRS-80, ROM stores the BASIC interpreter. EPROM could have been used as well in this application. Another use is to provide code conversions. For example, assume that you wish to make a code conversion between a certain code A and another code B. You could program the EPROM so that code A was the address to the EPROM, and code B was the corresponding data for that particular address. Another application is in the area of look-up tables for mathematical conversions. For example, the address could represent an angle, and the output data could be programmed to

U1—Intel 8255 PPI
U2—74LS138, 1 of 8 decoder
U3—7406, open collector inverter
Q1—2N3904 or equivalent
Q2,Q3—2N2907 or equivalent*
Red LED, Radio Shack cat #276-041 or equivalent
Yellow LED, Radio Shack cat #276-021 or equivalent
Green LED, Radio Shack cat #276-022 or equivalent
S1-DPDT Slide Switch, Radio Shack cat #275-407 or equivalent
Plug-in PC Board, Radio Shack cat #276-154 or 276-156
J1-dual 22-pin edge card socket, Radio Shack cat #276-1551 or equivalent
40-pin edge connector IDC type, Radio Shack cat #276-1558 or equivalent
Cable-flat ribbon, 40-conductor, Radio Shack cat #278-771 or equivalent
Regulator type 7805, Radio Shack cat #276-1770 or equivalent
R1, R8 10k 1/4W 5%
R7, R9, R11 3.3k 1/4W 5%
R13, R14 4.7k 1/4W 5%
R4 470 Ohm 1/4W 5%
R2, R5, R6 1k 1/2W 5%
R12 330 1/4W 5%
R3 5k pot
R10 2.7k 1/4W 5%
C1 10 uf, 10v electrolytic capacitor
C2, C3 – .1uf, 50v disc ceramic capacitor

*If you substitute for this device, make sure that the replacement part has low Vce Sat and a minimum HFE of 100.

Table 1. *Programmer parts*

T1 24 (OR 25.2) VAC CT @ 500 MA Min, Radio Shack cat #273-1512 or equivalent
C4 470 uf minimum @ 40V or greater, Radio Shack cat #272-1046 or equivalent
C6,C7 .1 uf 50V disc ceramic capacitors
C8 22 uf 20V electrolytic capacitor
C5 2000 uf minimum @ 25V or greater, Radio Shack cat #272-1020 or equivalent
Heat sink, TO220 type, Radio Shack cat #276-1363 or equivalent
D1 through D4-1A min, 100 PIV or greater, diodes, Radio Shack cat #276-1103 or equivalent
Fuse 1/2A slow blow type
Regulator type 7805, Radio Shack cat #276-1770 or equivalent
S2 SPST switch, Radio Shack cat #275-612 or equivalent
R15 100 Ohm potentiometer
R16 2.2k 1/4W 5%

Table 2. *Power supply parts*

represent the sine of that angle. Another use might be in the area of code acceptance. You could program the EPROM to accept only certain codes at its address lines, in which case the outputs would be programmed to give a certain output for a given valid input. It could also be programmed to yield zeros for all other invalid inputs. Imagine the amount of hardware that it would take to perform any of the above applications and you can see some of the power of the EPROM.

The Hardware

Figure 1 is the schematic of the programmer. The circuitry is quite simple; the entire circuit can be wire wrapped on a Radio Shack plug-in PC board using wire-wrap sockets. Figure 2 shows a layout of the major components of the circuit. For the small, discrete components, I suggest that you mount them on the board using VECTOR T-44 wire-wrap pins. This allows you to wire wrap them to the rest of the circuit without any need for soldering. You can obtain most of the components at nominal prices from advertisers in computer magazines and other related publications. If you have problems finding any of the parts, check your local Radio Shack store. If you prefer a ready-made printed circuit board for this project, there is one available that I will explain later in the text.

I chose to use the 8255 PPI as the heart of the circuit. The 8255 is a programmable I/O device designed for use with the 8080 microprocessor. Even though this device has been around for some time, it is quite powerful and tailor-made for the application I had in mind.

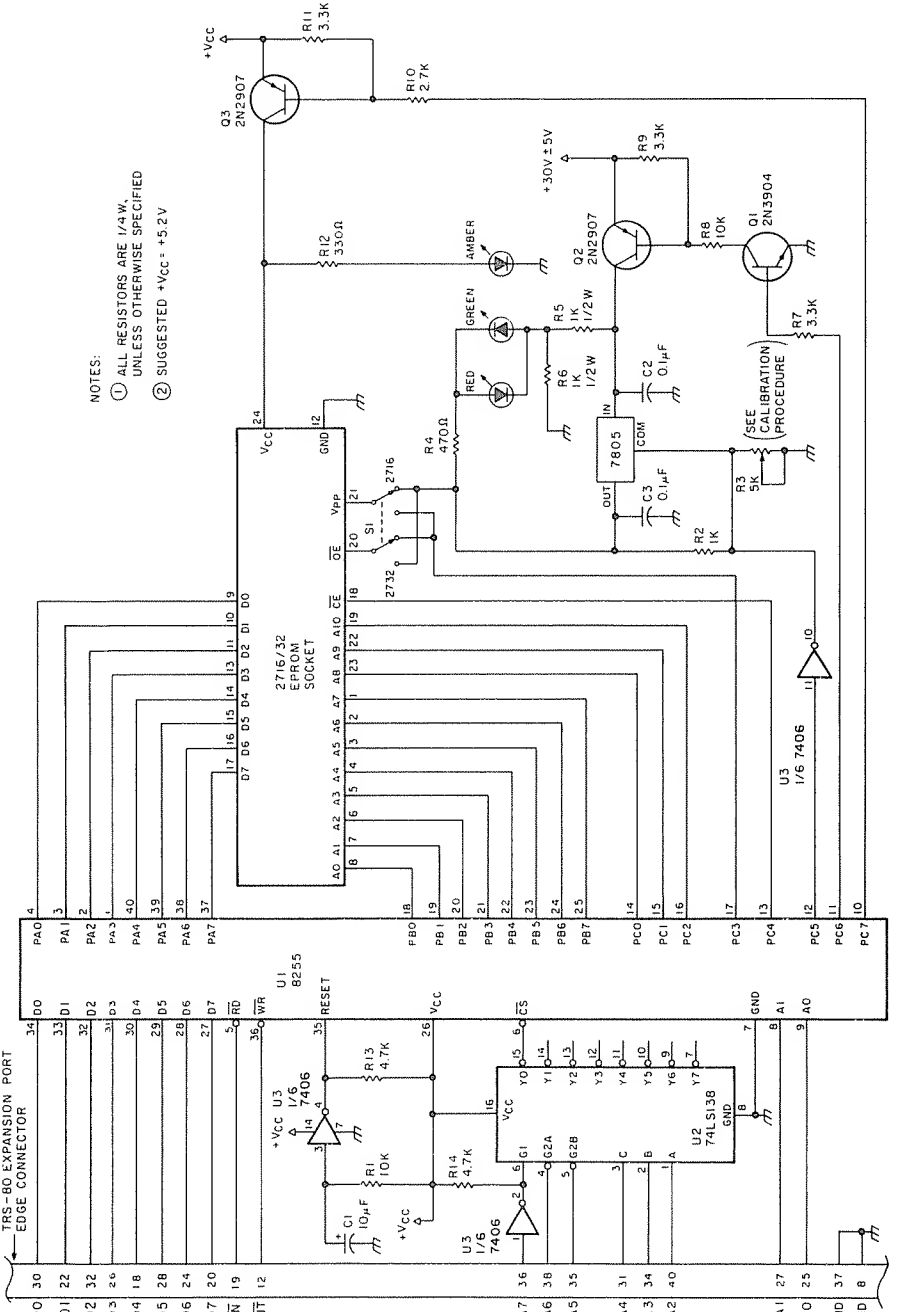


Figure 1. *Schematic of the EPROM programmer*

The 8255 is utilized in what the manufacturer refers to as mode 0 which allows any of the three addressable ports to be either input or output ports. The 74LS138 chip forms a decoder network which decodes the PIA ports. Port 3 controls the internal mode register which configures the device according to the codes it receives from the software. Once configured, the particular ports can then be addressed individually. An instruction to port 0 addresses port A. Instructions to ports 1 and 2 address ports B and C respectively. In this application, all three ports are used as outputs except when data is being read from the EPROM for verification. Port B and the four least

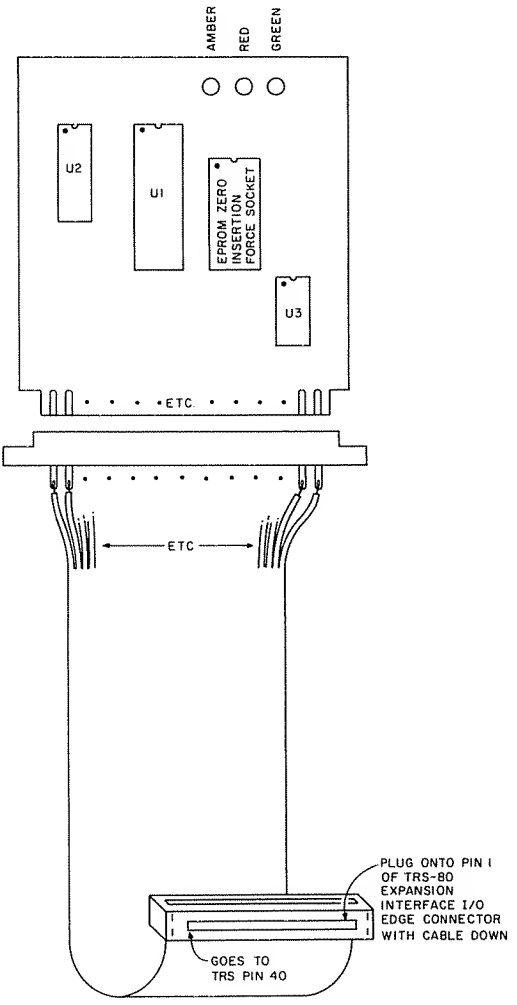


Figure 2. *Major components of the circuit*

significant bits of port C provide the addresses to the EPROM. The four MSBs of port C are control bits which activate the output enable bit of the EPROM and control the application of the various voltages to the EPROM. Port A is used bidirectionally to provide the data to the EPROM in one instance, and to read the data from the EPROM in the other. The software controls each of the three ports individually. Once the addressed port has been instructed to be an output, it latches the data presented on the data bus and holds it until it is instructed to do otherwise.

R1 and C1, together with inverter U3, comprise a "power on" reset circuit which ensures that all the ports in the 8255 go to the input mode when power is first applied to the programmer. In this mode, the 8255 outputs go to a high impedance state. This condition ensures that Q1, Q2, and Q3 go to the off state, inhibiting any voltages to the EPROM.

Transistor Q3 is a switch which applies + 5 volts to the EPROM. Transistor Q2 applies unregulated voltage to the regulator circuit which consists of the 7805 regulator, R2, and R3. Open collector inverter U3 pin 10 controls the voltage to the programming pin on the EPROM. With pin 11 low, the output voltage of the regulator is + 25 volts, the voltage required by the EPROM during the programming phase. The ratio of resistors R2 and R3 establishes this voltage. When a logic high is applied to U3 pin 11, the output voltage of the regulator drops to + 5 volts, the voltage required by the Vpp input of the 2716 EPROM at all times except during programming operations. The 2732 is different in that it requires a low level at this input except during programming. The software automatically applies the correct voltages to the EPROM during the various phases of programming and testing.

I have included three different colored LEDs which turn on and off at the appropriate times to let you know the status of the voltages that are being applied. This turns out to be a very nice feature in the initial phases of the circuit checkout. Each LED has a different significance. When + 25 volts is applied to the programming pin of the EPROM, the red LED lights. When the voltage at the programming pin goes to + 5 volts for purposes of reading or verification of the EPROM, the green LED lights. The amber LED signifies that power is applied to the EPROM. Before inserting any EPROM into the programming socket, make sure that no LEDs are lit. If any LED is on, recycle power to the programmer to ensure that all voltages to the EPROM are off. Under normal circumstances, all voltages are removed from the EPROM when the program is not running, making it safe to insert or remove an EPROM from the programming socket.

When working with the TRS-80 expansion bus, keep in mind that it is possible to damage circuitry within the computer if you inadvertently short-out or apply any voltages in the wrong places. Double-check all wiring to the circuit before you connect it to the computer. If you wire the circuit properly, computer operation is not affected in any way. In fact, you can

leave the circuit permanently attached if you wish. To avoid hand-wiring errors in such a touchy area, I have had printed circuit boards made for this project. The boards are available from The CO-ALERT Corporation, 12762 La Brida, Chino, California 91710 for $19.95 plus $2.50 for shipping and handling.

To couple my programmer to the expansion port of my interface, I use a flat ribbon cable of about three feet in length. When wiring this particular section of circuitry, refer to page 228 of your Level I manual for the function assignments of the 40-pin edge connector. The configuration of the 40-pin expansion interface connector is the same as that shown for the keyboard. A word of caution here: The pin numbers of the 40-pin IDC edge connector may not match those of the edge connector on the expansion interface PC board. Instead of relying on pin numbers, make sure that the functions get to the proper places by performing a continuity test of the circuit after the cable is fully wired. This is another reason I had PC boards made for this project. In the PC version, this problem is inherently resolved. LNW owners, be advised that your edge connector is inverted from the Radio Shack expansion interface. I use an LNW expansion interface and have experienced no problems with my programmer; in fact, it is permanently attached to my computer. I have successfully programmed EPROMs using a five-foot cable but I recommend that you use a cable just long enough to do the job.

The Power Supply

Figure 3 is a schematic of the power supply. I used a common transformer to keep the cost down. This circuit allows you to generate both voltages required by the programmer. A simple TO-220 heat sink on the regulator should suffice since the circuit current drain is low.

Hardware Checkout

After you have completed the construction of the circuit, perform the tests in this section to ensure that everything is operational. Do not connect the power supply to the programmer until you have successfully completed each test. Using a voltmeter, check the power supply Vcc voltage and adjust R15 until the voltage reads +5.2 volts. Next, check the unregulated voltage. This voltage should read about +35 volts, depending on the transformer you use. After the power supply tests, you can connect the power supply to the programmer. Using the test procedure below, you should be able to exercise all phases of the programmer.

With the power supply off, connect the programmer's cable connector to the 40-pin I/O edge connector on the expansion interface. Make sure the cable is oriented downward as shown on the mechanical drawing. Next, power-up the programmer. Go into Disk BASIC and issue the following

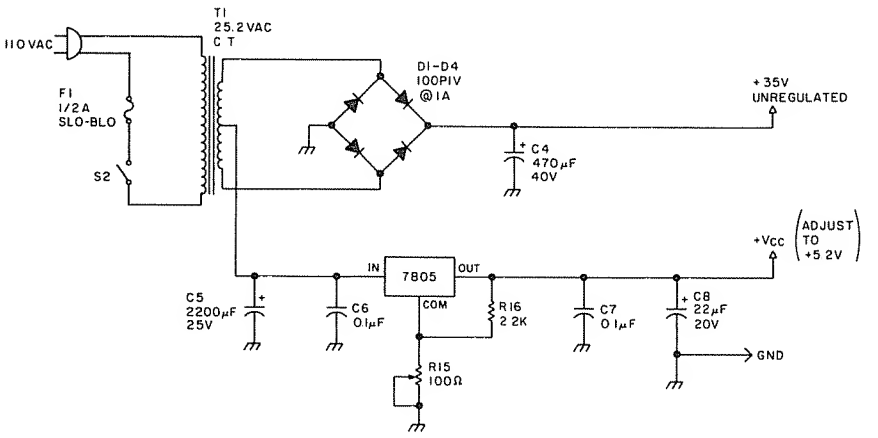


Figure 3. *Schematic of the power supply*

command: OUT 3, 128. This configures all ports for output. Next, enter OUT 2,64. This sends + 35 volts as input to the programmer's 7805. Adjust potentiometer R3 to yield + 25 volts on the output of the regulator. At this time, you may wish to apply some fingernail polish on the potentiometer to prevent accidental bumping of the adjustment. The red and amber LEDs should be lighted. Next, verify that + 5 volts appears on pin 24 of the EPROM socket.

You are now ready for the I/O check-out. Enter the following:

```
N = 0
OUT 0,N
OUT 1,N
OUT 2,N
```

Verify that all 8255 outputs are low. Repeat these OUT commands after entering N = 255, and the outputs should be high. The last test is to run the following BASIC source code which puts the LEDs in the following sequence: amber; amber/red; amber/green; all off; red; green; repeat.

```
10 OUT 3,128
20 FOR Z = 0 TO 255
30 OUT 2,Z
40 NEXT Z
50 GOTO 10
```

If your circuit successfully passes these tests, then it is functioning properly.

The Software

As you can see from the Program Listings, the software is written in assembly language. I added comments on the source programs to aid in understanding the program flow. I could have written it in BASIC, but ex-

ecution time would have been much longer. Using this software, a 2716 EPROM programs in less than 110 seconds. The 2732 takes approximately twice as long, as it contains twice the amount of memory.

I broke the software into four programs, each designed to perform a specific task. The READPROM program (Program Listing 1) is designed to allow you to read an EPROM and either save the contents on disk or transfer the data to another EPROM. The CHKERASE program (Program Listing 2) allows you to check an EPROM to see that it is completely erased. I highly recommend that you perform this test on every EPROM before you program it. The PGMEPROM program (Program Listing 3) is the software that does the actual programming of the device. The VERIFY program (Program Listing 4) performs verification of the programmed data. I found that structuring the software in this manner allows me considerably more flexibility than lumping it into a single program.

These programs are designed to reside in high memory to allow you to protect this section of memory in case you wish to combine any BASIC programs with this system. From BASIC, you have the opportunity to protect this section of memory to ensure that you don't destroy any data with the BASIC program. To accommodate those users with only 32K of memory, I protected memory above AA00H(43520 decimal). This leaves you room for up to 1K of special routines that could reside at the very top of memory.

To program the 2732 EPROM, you must make the following changes to the programs indicated:

READPROM

1) In line 130 change B3FFH to BBFFH.
2) Change line 220 to read:

```
LD  BC,4096
```

3) Insert line 275 and add the following comment:

```
;AND PLACE EPROM IN READ
```

4) Delete lines 290 and 300.
5) Change line 340 to read:

```
OR  20H
```

6) Delete line 350.

CHKERASE

1) Delete lines 370 and 380.
2) Change line 410 to read:

```
LD  BC,4096
```

3) Change line 450 to read:

```
OR  20H
```

PGMEPROM

1) Change line 150 to read:

```
BBFFH
```

2) Change line 340 to read:

```
LD  BC,4096
```

3) Change line 600 to read:

```
OR  40H
```

4) Change line 660 to read:

```
LD  A,50H
```

5) Delete line 670.
6) Change line 690 to read:

```
;AND PROGRAM PULSE GOES HI
```

VERIFY

1) In line 150, change B3FFH to read BBFFH.
2) Delete lines 360 and 370.
3) Change line 410 to read:

```
LD  BC,4096
```

4) Change line 450 to read:

```
OR  20H
```

These changes are easy to make if you load the original programs into EDTASM and make the changes. When the changes are complete, reassemble the programs and give them different names to distinguish them from the 2716 programs.

Saving Programs to Disk

After you have saved the source files and have generated object files using EDTASM, do a disk dump of the programs. The procedure for doing this appears in your DOS manual, but I will repeat it here. From DOS, load the READPROM object file, then perform the disk dump as per the example that follows. Give the program the name READPROM/CMD. While in DOS, type:

```
DUMP READPROM/CMD (START = X'AA00',END = X'AA50',TRA = X'AA00')
```

Now press ENTER, and the program goes to disk under the name of READPROM/CMD. Repeat this procedure for the remaining programs, giving each its appropriate name, followed by the extension /CMD. Once the programs have been saved to disk in this manner, they will execute directly from DOS. The ending addresses for the CHKERASE, PGMEPROM, and VERIFY programs are ABC0, AB08, and ABC7 respectively. These ending addresses are for the 2716 programs. The ending addresses for the 2732 are

AA4C, ABBC, AB07, and ABC3 for the READPROM, CHKERASE, PGMEPROM, and VERIFY programs respectively.

Executing Programs from DEBUG

I prefer to use DEBUG to execute the programs because it allows me considerable flexibility in observing program execution. I can do an investigation of RAM very easily to see how things have progressed. I can also jump to various portions of the program as I desire. To execute the program from DEBUG, proceed per the instructions that follow. The first order of business is to load the desired program from DOS. Next, type DEBUG and press ENTER. Press the BREAK key, and you enable the DEBUG program. Once in DEBUG, type GAA00. Press ENTER, and the program begins. After the program has completed execution, it returns to DOS. At this point, if you wish to make a memory dump, reenter DEBUG by pressing the BREAK key and type DAC00 followed by ENTER. If you wish to display an entire screen's worth of memory, press the S key. To observe the next 256 bytes of memory, press the + key. For a more detailed explanation of DEBUG functions, refer to the DOS manual under DEBUG.

Reading an EPROM

To read an EPROM that has been previously programmed, execute the READPROM program. This program reads the EPROM and places the contents in memory locations AC00H to B3FFH for the 2716 and locations AC00H to BBFFH for the 2732. Once at this location, the contents can then be copied to another EPROM or saved to disk through a disk dump.

Programming an EPROM

Before you program an EPROM, make absolutely sure that the 2716/2732 switch is in the proper position for the EPROM you are programming. Failure to do this will surely ruin the device. You also must ensure that the data to be programmed is residing in RAM locations AC00H through B3FFH for the 2716 or from AC00H through BBFFH for the 2732. You can do this by several different means. One method, already described, is reading a previously programmed EPROM using the READPROM program. This automatically places the data into the proper RAM locations for programming. Another method is to manually insert the data to be programmed into these RAM locations. You can do this using the M (modify memory) command of DEBUG. Another method is to POKE the data into the aforementioned RAM locations from a BASIC program. If you use this method, BASIC normally requires that you use decimal notation and not hexadecimal for POKEing into memory. There is a little trick that you can use, however, to get around this problem and save yourself a lot of hassle converting from one number system to another. The following BASIC pro-

gram is an example of how this is done.

```
10 N = &HAC00 :REM (HEREIN LIES THE TRICK:&HAC00 = Decimal 4032)
20 FOR X = 0 TO 8
30 FOR I = 0 to 255
40 POKE N,I
50 N = N + 1
60 NEXT I
70 NEXT X
80 END
```

This program alternately places the number sequence from 0 to 255 eight times in the 2716. The program itself is not very useful other than to illustrate a point, although you might consider using it to program your first EPROM. Remember that you can always erase it and use it again. It allows you to exercise the programmer fully and has an easily recognizable data pattern.

I highly recommend that you run the CHKERASE program before you start to program an EPROM. This program checks all locations of the EPROM to see that they are completely erased. If any data other than FF hex is found at any location, the program stops and tells you that data exists at the address specified. You are given the option to check the EPROM for other faults or to terminate the operation. If no faults are found, the program continues without interruption. If a fault occurs, follow the program prompts. This ensures that the program will remove all EPROM voltages at the completion of the test.

Once you are sure the EPROM is fully erased, execute the PGMEPROM program. This program does the actual programming of the EPROM. While it is running, the red LED is on, indicating that + 25 volts is present on the programming pin of the EPROM. The address of the location that is being programmed is displayed on the screen as well. This address is in hex notation. When the address reaches 800 hex (2048 decimal), the programming phase is complete for the 2716. For the 2732, the address goes to 1000 hex (4096 decimal). At the completion of the programming phase, the system returns to DOS. At this point, load and execute the VERIFY program to check the contents of the EPROM data you just programmed against the data that you used to program it. When verifying the 2716, the red LED is off and the green LED is on. This signifies that the programming voltage has been removed from the EPROM and that + 5 volts is now applied to the programming pin, placing the EPROM in the read mode. The 2732 is different in that the programming pin requires a low level at its input, causing the green LED to be off. If the EPROM fails to verify, the program stops and specifies the address location of the incorrect data, along with the actual data that was read. In addition, the program specifies the data that should have been read and gives you the option of checking for further failures or

ending the operation. If no failures are found, the program continues without interruption. This signifies that the EPROM was successfully programmed.

Using the Programmer as an I/O Port

This programmer can serve as an I/O port if you connect a 24-pin flat ribbon cable to the zero insertion socket, using a 24-pin dip socket. You can use 20 of the 24 pins of the EPROM socket from the 8255 for your special I/O applications. You must make the ground at pin 12 common to your I/O circuit. Instantly, you have two eight-bit bidirectional ports and one four-bit bidirectional port at your disposal. Use these ports for whatever application you wish. The only restriction is that you must limit each output to the equivalent of one TTL load and restrict all inputs to TTL voltage levels.

DATA (TO#3)	Port A (#0)	Port B (#1)	Port C lo (#2 LSBs)	Port C hi (#2 MSBs)
* 9BH	input	input	input	input
9AH	input	input	output	input
99H	input	output	input	input
98H	input	output	output	input
8BH	output	input	input	input
8AH	output	input	output	input
89H	output	output	input	input
88H	output	output	output	input

* = automatic configuration at power-on reset

Table 3. *I/O configuration*

You can use the BASIC OUT and IN instructions to control the 8255. See Table 3 for information on configuring the I/O port to your needs. First, set the mode register according to the way you wish to configure your ports. The first column in the table gives you the hex number that must be sent to the internal mode register to configure it per the associated row. For example, if you wished to make ports A and C inputs and port B an output, you would go into Disk BASIC and type the following:

```
OUT 3,&H99
```

This command to the internal mode register configures the ports as described above. You can now either output data to a port or input data from a port as described in the following examples.

- Input of data is done as follows:

```
A = INP(0)
PRINT A
```

- Output of data is done as follows:

```
OUT 1,nn
```

Note that nn can be any decimal number between 0 and 255.

The simplicity of this project results from sharing the hardware and software responsibilities. It offers both hardware and software oriented individuals exposure to both worlds.

Program Listing 1. *READPROM*

```
                00100 ;*******************************************************
                00110 ;*           >>>>>>READPROM<<<<<<                       *
                00120 ;* THIS PROGRAM WILL READ AN EPROM AND PUT THE CONTENTS *
                00130 ;* IN LOCATIONS AC00H TO B3FFH. USE DEBUG TO READ OUT   *
                00140 ;* DATA.                                               *
                00150 ;*                                                    *
                00160 ;*      PROGRAM WRITTEN BY ABEL J. TAPIA  06/15/81     *
                00170 ;*******************************************************
AA00            00180          ORG     0AA00H
AA00 CDC901     00190          CALL    01C9H           ;CLR SCREEN RTN
AA03 1100AC     00200          LD      DE,0AC00H       ;DATA DESTINATION
AA06 210000     00210          LD      HL,0000H        ;EPROM ADDRESS
AA09 010008     00220          LD      BC,2048         ;NO OF BYTES
AA0C 3E90       00230          LD      A,90H           ;PPI CONTROL WORD
                00240                                  ;TO MAKE A PORT=INPUT
                00250                                  ;B&C PORTS=OUTPUT
AA0E D303       00260          OUT     (03),A          ;SEND IT OUT
AA10 3E20       00270          LD      A,20H           ;CMD WRD TO TURN ON PWR
AA12 D302       00280          OUT     (02),A          ;SEND TO PORT C
AA14 3E60       00290          LD      A,60H           ;CMD WRD FOR EPROM READ
AA16 D302       00300          OUT     (02),A          ;SEND IT OUT
AA18 7D         00310 CONT     LD      A,L             ;ADDRESS LSB
AA19 D301       00320          OUT     (01),A          ;SEND TO B PORT
AA1B 7C         00330          LD      A,H             ;ADDRESS MSB
AA1C F660       00340          OR      60H             ;CONTROL WRD TO APPLY +5V
                00350                                  ;TO EPROM VPP INPUT TO
                00360                                  ;ALLOW EPROM TO READ
AA1E D302       00370          OUT     (02),A          ;SEND WORD TO PORT C
AA20 DB00       00380          IN      A,(00)          ;READ EPROM DATA-PORT A
AA22 12         00390          LD      (DE),A          ;SEND TO DESTINATION
AA23 0B         00400          DEC     BC              ;DECREMENT BYTE COUNTER
AA24 78         00410          LD      A,B             ;PUT MSB BYTE IN A
AA25 B1         00420          OR      C               ;OR W/LSB BYTE
AA26 CA2EAA     00430          JP      Z,FINISH        ;ALL BYTES PROCESSED?
AA29 13         00440          INC     DE              ;INC DESTINATION PTR
AA2A 23         00450          INC     HL              ;INC EPROM ADDRESS
AA2B C318AA     00460          JP      CONT            ;CONTINUE PROCESS
AA2E 11143E     00470 FINISH   LD      DE,3C00H+532    ;SCREEN POINTER
AA31 010D00     00480          LD      BC,13           ;NO OF CHARS
AA34 2144AA     00490          LD      HL,MSG          ;PT TO MESSAGE
AA37 EDB0       00500          LDIR                    ;SEND IT OUT
AA39 3E9A       00510          LD      A,9AH           ;CMD WRD TO MAKE
                00520                                  ;A&B PORTS=IN,C=OUT
AA3B D303       00530          OUT     (03),A          ;SEND IT OUT
AA3D 3EA0       00540          LD      A,0A0H          ;CMD WRD TO TURN OFF PWR
                00550                                  ;TO EPROM
AA3F D302       00560          OUT     (02),A          ;SEND OUT CNTRL WORD
AA41 C32D40     00570          JP      402DH           ;RETURN TO DOS
AA44 54         00580 MSG      DEFM    'TEST COMPLETE'
0000            00590          END
00000 TOTAL ERRORS
```

Program Listing 2. *CHKERASE*

```
                00100 ;*******************************************************
                00110 ;*           >>>>>>CHKERASE<<<<<<                      *
                00120 ;*                                                     *
                00130 ;* THIS PROGRAM WILL TEST EPROM TO ENSURE THAT ALL     *
                00140 ;* LOCATIONS ARE ERASED. IF EPROM CONTAINS DATA,PROGRAM *
                00150 ;* WILL STOP AND DISPLAY ADDRESS OF LOCATION CONTAINING *
                00160 ;* THE DATA.                                           *
                00170 ;*                                                     *
                00180 ;*    PROGRAM WRITTEN BY ABEL J. TAPIA  06/30/81       *
                00190 ;*******************************************************
AA00            00200          ORG     0AA00H
```

Program continued

```
3C00            00210 VIDEO   EQU   3C00H          ;START OF SCREEN
AA00 00         00220 SAVEA   DEFB  0              ;SAVE A REG RAM LOC
AA01 0000       00230 SAVEBC  DEFW  00             ;SAVE BC REG PTR RAM LOC
AA03 0000       00240 SAVEDE  DEFW  00             ;SAVE DE REG PTR RAM LOC
AA05 0000       00250 SAVEHL  DEFW  00             ;SAVE HL REG PTR RAM LOC
AA07 CDC901     00260         CALL  01C9H          ;CLR SCREEN ROUTINE
AA0A 3E90       00270         LD    A,90H          ;CONTROL WORD TO PPI
                00280                              ;TO MAKE PORT A=IN
                00290                              ;PORTS B&C=OUT
AA0C D303       00300         OUT   (03),A         ;SEND IT OUT
AA0E 3E20       00310         LD    A,20H          ;CMD WRD FOR PWR ON
AA10 D302       00320         OUT   (02),A         ;TURN ON PWR TO EPROM
AA12 11CC3D     00330         LD    DE,VIDEO+460   ;SCREEN POINTER
AA15 2101AB     00340         LD    HL,MSG1        ;POINT TO CHECK MESSAGE
AA18 012800     00350         LD    BC,40          ;NO. OF CHARS
AA1B EDB0       00360         LDIR                 ;SEND OUT MESSAGE
AA1D 3E60       00370         LD    A,60H          ;CMD WRD FOR EPROM READ
AA1F D302       00380         OUT   (02),A         ;SEND IT OUT
AA21 CDCDAA     00390         CALL  WAIT           ;DISPLAY WAIT ROUTINE
AA24 110000     00400         LD    DE,0000H       ;EPROM START ADDRESS
AA27 010008     00410         LD    BC,2048        ;NO OF BYTES
AA2A 7B         00420 CONT    LD    A,E            ;EPROM ADDRESS LSB
AA2B D301       00430         OUT   (01),A         ;SEND TO B PORT
AA2D 7A         00440         LD    A,D            ;EPROM ADDRESS MSB
AA2E F660       00450         OR    60H            ;CONTROL BITS FOR READ
AA30 D302       00460         OUT   (02),A         ;SEND TO C PORT LO & HI
AA32 DB00       00470         IN    A,(00)         ;READ EPROM DATA-PORT A
AA34 FEFF       00480         CP    0FFH           ;IS IT ERASED?
AA36 C259AA     00490         JP    NZ,ERROR       ;IF NOT GOTO ERROR RTN
AA39 0B         00500 TSTAGN  DEC   BC             ;DECREMENT BYTE COUNTER
AA3A 78         00510         LD    A,B            ;PUT MSB BYTE IN A
AA3B B1         00520         OR    C              ;OR W/LSB BYTE
AA3C CA43AA     00530         JP    Z,FINISH       ;ALL BYTES PROCESSED?
AA3F 13         00540         INC   DE             ;INCR EPROM ADDRESS
AA40 C32AAA     00550         JP    CONT           ;CONTINUE PROCESS
AA43 F5         00560 FINISH  PUSH  AF             ;SAVE A
AA44 CDC901     00570         CALL  01C9H          ;CLR SCREEN
AA47 F1         00580         POP   AF             ;RECALL A
AA48 11083E     00590         LD    DE,VIDEO+520   ;SCREEN POINTER
AA4B 013300     00600         LD    BC,51          ;NO OF CHARS
AA4E 2129AB     00610         LD    HL,MSG2        ;POINT TO MESSAGE
AA 1 EDB0       00620         LDIR                 ;SEND IT OUT
AA53 CDCDAA     00630         CALL  WAIT           ;DISPLAY WAIT ROUTINE
AA56 C3D7AA     00640         JP    DOS            ;GO BACK TO DOS
AA59 ED4301AA   00650 ERROR   LD    (SAVEBC),BC    ;SAVE BYTE COUNT
AA5D ED5303AA   00660         LD    (SAVEDE),DE    ;SAVE ADDR POINTER
AA61 3200AA     00670         LD    (SAVEA),A      ;SAVE EPROM DATA
AA64 CDC901     00680         CALL  01C9H          ;CLR SCREEN
AA67 11003E     00690         LD    DE,VIDEO+512   ;SCREEN POINTER
AA6A 013E00     00700         LD    BC,62          ;NO. OF CHARS
AA6D 215CAB     00710         LD    HL,MSG3        ;ERROR MESSAGE
AA70 EDB0       00720         LDIR                 ;PRINT ERROR MESSAGE
AA72 ED5B03AA   00730         LD    DE,(SAVEDE)    ;RECALL ADDR POINTER
AA76 EB         00740         EX    DE,HL          ;PUT EPROM ADDR IN HL
AA77 7C         00750         LD    A,H            ;EPROM ADDRESS MSB
AA78 CDE2AA     00760         CALL  CNVASC         ;GOTO ASCII ROUTINE
AA7B 7C         00770         LD    A,H            ;GET EPROM ADDR 1ST DIGIT
AA7C 321F3E     00780         LD    (VIDEO+543),A  ;ADDR 1ST DIGIT TO SCREEN
AA7F 7D         00790         LD    A,L            ;EPROM ADDR 2ND DIGIT
AA80 32203E     00800         LD    (VIDEO+544),A  ;ADDR 2ND DIGIT TO SCREEN
AA83 ED5B03AA   00810         LD    DE,(SAVEDE)    ;RECALL ADDR POINTER
AA87 EB         00820         EX    DE,HL          ;PUT EPROM LSB ADDR IN HL
AA88 7D         00830         LD    A,L            ;EPROM ADDR LSB
AA89 CDE2AA     00840         CALL  CNVASC         ;CONV TO ASCII
AA8C 7C         00850         LD    A,H            ;GET ASCII 3RD DIGIT
AA8D 32213E     00860         LD    (VIDEO+545),A  ;ADDR 3RD DIGIT TO SCREEN
AA90 7D         00870         LD    A,L            ;GET ASCII 4TH DIGIT
AA91 32223E     00880         LD    (VIDEO+546),A  ;ADDR 4TH DIGIT TO SCREEN
AA94 3A00AA     00890         LD    A,(SAVEA)      ;RECALL EPROM DATA
AA97 CDE2AA     00900         CALL  CNVASC         ;CONV TO ASCII
```

```
AA9A 7C       00910          LD      A,H             ;GET DATA 1ST ASCII CHAR
AA9B 322C3E   00920          LD      (VIDEO+556),A   ;OUT EPROM DATA 1ST DIGIT
AA9E 7D       00930          LD      A,L             ;GET DATA 2ND ASCII CHAR
AA9F 322D3E   00940          LD      (VIDEO+557),A   ;OUT EPROM DATA 2ND DIGIT
AAA2 11873E   00950          LD      DE,VIDEO+647    ;SCREEN POINTER
AAA5 219AAB   00960          LD      HL,MSG4         ;RETEST MESSAGE
AAA8 012700   00970          LD      BC,39           ;NO OF CHARS
AAAB EDB0     00980          LDIR                    ;SEND  MESSAGE TO SCREEN
              00990 ;
              01000 ;---------KEYBOARD SCAN ROUTINE-------------------------
              01010 ;
AAAD CD4900   01020 KYBRD    CALL    0049H           ;CALL KEYBOARD SCAN RTN
AAB0 FE59     01030          CP      59H             ;IS INPUT CHAR=Y?
AAB2 CABCAA   01040          JP      Z,RESTOR        ;IF SO,RESTORE ALL REGS
AAB5 FE4E     01050          CP      4EH             ;IS CHAR=N?
AAB7 CAD7AA   01060          JP      Z,DOS           ;IF SO, END TEST
AABA 18F1     01070          JR      KYBRD           ;KEEP SCANNING
              01080 ;-------------------------------------------------------
AABC 3A00AA   01090 RESTOR   LD      A,(SAVEA)       ;RESTORE
AABF ED4B01AA 01100          LD      BC,(SAVEBC)     ;  ALL
AAC3 ED5B03AA 01110          LD      DE,(SAVEDE)     ;     REGISTERS
AAC7 2A05AA   01120          LD      HL,(SAVEHL)     ;           AND:
AACA C339AA   01130          JP      TSTAGN          ;CONTINUE TESTING
              01140 ;
              01150 ;-----------DISPLAY TIMER ROUTINE-----------------------
              01160 ;
AACD 0100FA   01170 WAIT     LD      BC,64000
AAD0 0B       01180 WAIT1    DEC     BC
AAD1 78       01190          LD      A,B
AAD2 B1       01200          OR      C
AAD3 C2D0AA   01210          JP      NZ,WAIT1
AAD6 C9       01220          RET
              01230 ;-------------------------------------------------------
AAD7 3E9A     01240 DOS      LD      A,9AH           ;CMD WRD TO MAKE
              01250                                  ;A&B PORTS=IN, C=OUT
AAD9 D303     01260          OUT     (03),A          ;SEND IT OUT
AADB 3EA0     01270          LD      A,0A0H          ;PREPARE FOR PWR OFF
AADD D302     01280          OUT     (02),A          ;TURN OFF PWR TO EPROM
AADF C32D40   01290          JP      402DH           ;GO BACK TO DOS
              01300 ;
              01310 ;----------CONVERT TO ASCII ROUTINE----------------------
              01320 ;
AAE2 4F       01330 CNVASC   LD      C,A             ;SAVE HEX DIGITS
AAE3 CB3F     01340          SRL     A               ;ALIGN HI DIGIT
AAE5 CB3F     01350          SRL     A
AAE7 CB3F     01360          SRL     A
AAE9 CB3F     01370          SRL     A
AAEB CDF7AA   01380          CALL    TEST            ;CNV TO ASCII
AAEE 67       01390          LD      H,A             ;SAVE FOR RETURN
AAEF 79       01400          LD      A,C             ;RESTORE ORIGINAL
AAF0 E60F     01410          AND     0FH             ;GET LO DIGIT
AAF2 CDF7AA   01420          CALL    TEST            ;CONV TO ASCII
AAF5 6F       01430          LD      L,A             ;SAVE FOR RETURN
AAF6 C9       01440          RET
AAF7 C630     01450 TEST     ADD     A,30H           ;CONVERSION FACTOR
AAF9 FE3A     01460          CP      3AH             ;TEST FOR 0-9
AAFB FA00AB   01470          JP      M,TEST1         ;GO IF 0-9
AAFE C607     01480          ADD     A,7             ;ELSE CORRECT FOR A-F
AB00 C9       01490 TEST1    RET                     ;RETURN
              01500 ;-------------------------------------------------------
AB01 43       01510 MSG1     DEFM    'CHECKING TO ENSURE EPROM IS FULLY ERASED'
AB29 54       01520 MSG2     DEFM    'TEST COMPLETE..OK TO PROGRAM IF NO ERRORS DISPLAYED'
AB5C 45       01530 MSG3     DEFM    'EPROM CONTAINS DATA AT ADDRESS:    H..READS:  ..SHOULD READ F
AB9A 44       01540 MSG4     DEFM    'DO YOU WISH TO CONTINUE TESTING ? (Y/N)'
0000          01550          END
00000 TOTAL ERRORS
```

Program Listing 3. *PGMEPROM*

```
                00100 ;*************************************************
                00110 ;*           >>>>>>PGMEPROM<<<<<<                 *
                00120 ;*                                               *
                00130 ;* THIS SOFTWARE WILL PROGRAM THE EPROM USING     *
                00140 ;* PROGRAM DATA RESIDING IN RAM LOCATIONS ACOOH TO *
                00150 ;* B3FFH.                                         *
                00160 ;*                                               *
                00170 ;*    PROGRAM WRITTEN BY ABEL J. TAPIA  06/06/81 *
                00180 ;*************************************************
AA00            00190          ORG     0AA00H          ;PROGRAM START
AC00            00200 DATA     EQU     0AC00H          ;STARTING LOC OF RAM DATA
3E12            00210 VIDEO    EQU     3C00H+530       ;START PT OF MSG ON SCRN
AA00 3E80       00220          LD      A,80H           ;MODE 0 CNTRL WRD FOR PPI
                00230                                  ;FOR A,B,C PORTS=OUTPUT
AA02 D303       00240          OUT     (03),A          ;WRD TO CNTRL REG IN PPI
AA04 3E20       00250          LD      A,20H           ;CMD FOR EPROM PWR ON
                00260                                  ;ALL OTHER VOLTAGES OFF
AA06 D302       00270          OUT     (02),A          ;WORD OUT TO PORT C
AA08 CDC901     00280          CALL    01C9H           ;CLR SCREEN ROUTINE
AA0B 015900     00290          LD      BC,89D          ;NO. OF CHARS
AA0E 11923D     00300          LD      DE,VIDEO-128    ;SCREEN POINTER
AA11 219CAA     00310          LD      HL,MSG1         ;POINT TO MESSAGE 1
AA14 EDB0       00320          LDIR                    ;XFER MESSAGE
AA16 110000     00330          LD      DE,0000         ;EPROM ADDRESS START
AA19 010008     00340          LD      BC,2048         ;TOTAL NO. OF BYTES
AA1C 2100AC     00350          LD      HL,DATA         ;POINT TO DATA
AA1F C5         00360 CONT     PUSH    BC              ;SAVE BYTE COUNT
AA20 7E         00370          LD      A,(HL)          ;GET PROGRAMMING DATA
AA21 D300       00380          OUT     (00),A          ;SEND DATA TO EPROM
                00390                                  ;VIA PORT A
AA23 7B         00400          LD      A,E             ;GET EPROM ADDR LSB
AA24 D301       00410          OUT     (01),A          ;OUT TO EPROM VIA PORT B
AA26 F5         00420          PUSH    AF              ;SAVE A
AA27 E5         00430          PUSH    HL              ;SAVE HL
AA28 CD7DAA     00440          CALL    CNVASC          ;CONV TO ASCII
AA2B 7D         00450          LD      A,L             ;GET ASCII LSB
AA2C 32E83D     00460          LD      (VIDEO-42),A    ;SEND IT TO SCREEN
AA2F 7C         00470          LD      A,H             ;GET ASCII MSB
AA30 32E73D     00480          LD      (VIDEO-43),A    ;SEND IT TO SCREEN
AA33 E1         00490          POP     HL              ;RESTORE HL
AA34 F1         00500          POP     AF              ;RESTORE A
AA35 7A         00510          LD      A,D             ;GET EPROM ADDR MSB
AA36 E60F       00520          AND     0FH             ;MASK 4 MSBS TO=PORT C LO
AA38 F5         00530          PUSH    AF              ;SAVE A
AA39 E5         00540          PUSH    HL              ;SAVE HL
AA3A CD7DAA     00550          CALL    CNVASC
AA3D 7D         00560          LD      A,L             ;GET ASCII MSB DIGIT
AA3E 32E63D     00570          LD      (VIDEO-44),A    ;SEND IT TO SCREEN
AA41 E1         00580          POP     HL              ;RESTORE HL
AA42 F1         00590          POP     AF              ;RESTORE A
AA43 F658       00600          OR      58H             ;CMD WRD FOR PRGM START
                00610                                  ;COMBINE W/ADDR MSB IN
                00620                                  ;PORT C
AA45 D302       00630          OUT     (02),A          ;SEND OUT PORT C WORD
                00640                                  ;TO START PROGRAMMING
AA47 CD62AA     00650          CALL    TIMER           ;PROGRAM PULSE= 50 MS
AA4A 3E48       00660          LD      A,48H           ;CTRL WRD TO ENSURE THAT:
                00670                                  ;O/E BIT REMAINS HI
                00680                                  ;THAT +25V STAYS ON
                00690                                  ;AND PRGRM PULSE GOES LO
                00700                                  ;TO STOP PROGRAM PHASE
AA4C D302       00710          OUT     (02),A          ;WORD OUT TO PORT C
AA4E 13         00720          INC     DE              ;INCR EPROM ADDRESS
AA4F 23         00730          INC     HL              ;GET NEXT DATA BYTE
AA50 C1         00740          POP     BC              ;RECALL LATEST BYTE COUNT
AA51 0B         00750          DEC     BC              ;DECREMENT BYTE COUNTER
AA52 78         00760          LD      A,B             ;LOAD LSB IN A
```

```
AA53 B1       00770          OR      C               ;OR WITH MSB
AA54 C21FAA   00780          JP      NZ,CONT         ;KEEP GOING TILL DONE
AA57 3E9A     00790          LD      A,9AH           ;CMD WRD TO MAKE
              00800                                  ;A&B PORTS=IN,C=OUT
AA59 D303     00810          OUT     (03),A          ;SEND IT OUT
AA5B 3EA0     00820          LD      A,0A0H          ;CMD WRD TO SET PROGRAM
              00830                                  ;PULSE = TO LO
              00840                                  ;SET EPROM OUT OF PGM MDE
              00850                                  ;TURN OFF +25V & +5V PWR
AA5D D302     00860          OUT     (02),A          ;WORD OUT TO PORT C
AA5F C36CAA   00870          JP      FINISH          ;TEST COMPLETE ROUTINE
              00880 ;
              00890 ;---------------50MS TIMING LOOP-----------------------
              00900 ;
AA62 01650E   00910 TIMER    LD      BC,3685
AA65 0B       00920 TIMOUT   DEC     BC
AA66 78       00930          LD      A,B
AA67 B1       00940          OR      C
AA68 C265AA   00950          JP      NZ,TIMOUT
AA6B C9       00960          RET
              00970 ;-------------------------------------------------------
AA6C CDC901   00980 FINISH   CALL    01C9H           ;CLR SCREEN
AA6F 11123E   00990          LD      DE,VIDEO        ;POINT TO SCREEN
AA72 21BCAA   01000          LD      HL,MSG2         ;POINT TO MESSAGE 2
AA75 011400   01010          LD      BC,20           ;NO. OF CHARS
AA78 EDB0     01020          LDIR                    ;SEND OUT MESSAGE
AA7A C32D40   01030          JP      402DH           ;GOTO DOS
              01040 ;
              01050 ;---------------CONVERT TO ASCII ROUTINE---------------
              01060 ;
AA7D 4F       01070 CNVASC   LD      C,A
AA7E CB3F     01080          SRL     A
AA80 CB3F     01090          SRL     A
AA82 CB3F     01100          SRL     A
AA84 CB3F     01110          SRL     A
AA86 CD92AA   01120          CALL    TEST
AA89 67       01130          LD      H,A
AA8A 79       01140          LD      A,C
AA8B E60F     01150          AND     0FH
AA8D CD92AA   01160          CALL    TEST
AA90 6F       01170          LD      L,A
AA91 C9       01180          RET
AA92 C630     01190 TEST     ADD     A,30H
AA94 FE3A     01200          CP      3AH
AA96 FA9BAA   01210          JP      M,TEST1
AA99 C607     01220          ADD     A,7
AA9B C9       01230 TEST1    RET
              01240 ;-------------------------------------------------------
AA9C 45       01250 MSG1     DEFM    'EPROM PROGRAMMING IN PROGRESS...
                                      PROGRAMMING ADDRESS:    H'
AABC 50       01260 MSG2     DEFM    'PROGRAMMING COMPLETE'
0000          01270          END
00000 TOTAL ERRORS
```

Program Listing 4. *VERIFY*

```
              00100 ;*******************************************************
              00110 ;*                >>>>>>VERIFY<<<<<<                   *
              00120 ;* THIS PROGRAM WILL VERIFY AN  EPROM THAT HAS BEEN     *
              00130 ;* PROGRAMMED BY COMPARING THE OUTPUTS WITH THE DATA    *
              00140 ;* THAT WAS USED TO PROGRAM THE DEVICE. THE INPUT DATA  *
              00150 ;* RESIDES IN MEMORY LOCATIONS AC00H TO B3FFH.          *
              00160 ;*                                                     *
              00170 ;*    PROGRAM WRITTEN BY ABEL J. TAPIA     07/05/81     *
              00180 ;*******************************************************
AA00          00190          ORG     0AA00H
```

Program continued

```
3C00            00200 VIDEO   EQU     3C00H           ;START OF SCREEN
AA00 CDC901     00210         CALL    01C9H           ;CLR SCREEN ROUTINE
AA03 0000       00220 SAVEBC  DEFW    00              ;BYTE COUNT SAVE LOC
AA05 0000       00230 SAVEHL  DEFW    00              ;DATA PTR SAVE LOC
AA07 0000       00240 SAVEDE  DEFW    00              ;ADDR PTR SAVE LOC
AA09 00         00250 SAVEA   DEFB    00              ;EPROM DATA SAVE LOC
AA0A 3E90       00260         LD      A,90H           ;CONTROL WORD TO PP
                00270                                 ;TO MAKE A PORT=IN
                00280                                 ;B&C PORTS =OUT
AA0C D303       00290         OUT     (03),A          ;SEND IT OUT
AA0E 3E20       00300         LD      A,20H           ;CMD WRD TO TURN ON PWR
AA10 D302       00310         OUT     (02),A          ;TURN ON EPROM PWR
AA12 11D43D     00320         LD      DE,VIDEO+468    ;SCREEN POINTER
AA15 2120AB     00330         LD      HL,MSG1         ;POINT TO VERIFY MESSAGE
AA18 010F00     00340         LD      BC,15           ;NO. OF CHARS
AA1B EDB0       00350         LDIR                    ;SEND OUT MESSAGE
AA1D 3E60       00360         LD      A,60H           ;CMD WRD FOR EPROM READ
AA1F D302       00370         OUT     (02),A          ;SEND IT OUT
AA21 CDECAA     00380         CALL    WAIT            ;DISPLAY WAIT ROUTINE
AA24 2100AC     00390         LD      HL,0AC00H       ;IST INPUT DATA LOCATION
AA27 110000     00400         LD      DE,0000H        ;EPROM START ADDRESS
AA2A 010008     00410         LD      BC,2048         ;NO OF BYTES
AA2D 7B         00420 CONT    LD      A,E             ;EPROM ADDRESS LSB
AA2E D301       00430         OUT     (01),A          ;SEND TO B PORT
AA30 7A         00440         LD      A,D             ;EPROM ADDRESS MSB
AA31 F660       00450         OR      60H             ;CONTROL BITS FOR READ
AA33 D302       00460         OUT     (02),A          ;SEND TO C PORT LO & HI
AA35 DB00       00470         IN      A,(00)          ;READ EPROM DATA-PORT A
AA37 BE         00480         CP      (HL)            ;IS IT SAME AS INPUT?
AA38 C259AA     00490         JP      NZ,ERROR        ;IF NOT GOTO ERROR RTN
AA3B 0B         00500 TSTAGN  DEC     BC              ;DECREMENT BYTE COUNTER
AA3C 78         00510         LD      A,B             ;PUT MSB BYTE IN A
AA3D B1         00520         OR      C               ;OR W/LSB BYTE
AA3E CA46AA     00530         JP      Z,FINISH        ;ALL BYTES PROCESSED?
AA41 23         00540         INC     HL              ;INCR DATA POINTER
AA42 13         00550         INC     DE              ;INCR EPROM ADDRESS
AA43 C32DAA     00560         JP      CONT            ;CONTINUE PROCESS
AA46 F5         00570 FINISH  PUSH    AF              ;SAVE A
AA47 CDC901     00580         CALL    01C9H           ;CLR SCREEN
AA4A F1         00590         POP     AF              ;RECALL A
AA4B 11003E     00600         LD      DE,VIDEO+512     ;SCREEN POINTER
AA4E 013D00     00610         LD      BC,61           ;NO OF CHARS
AA51 212FAB     00620         LD      HL,MSG2         ;POINT TO MESSAGE
AA54 EDB0       00630         LDIR                    ;SEND IT OUT
AA56 C3F6AA     00640         JP      DOS             ;GO BACK TO DOS
AA59 ED4303AA   00650 ERROR   LD      (SAVEBC),BC     ;SAVE BYTE COUNT
AA5D ED5307AA   00660         LD      (SAVEDE),DE     ;SAVE ADDR PTR
AA61 2205AA     00670         LD      (SAVEHL),HL     ;SAVE DATA PTR
AA64 3209AA     00680         LD      (SAVEA),A       ;SAVE EPROM DATA
AA67 CDC901     00690         CALL    01C9H           ;CLR SCREEN
AA6A 11003E     00700         LD      DE,VIDEO+512    ;SCREEN POINTER
AA6D 013200     00710         LD      BC,50           ;NO. OF CHARS
AA70 216CAB     00720         LD      HL,MSG3         ;ERROR MESSAGE
AA73 EDB0       00730         LDIR                    ;PRINT ERROR MESSAGE
AA75 ED5B07AA   00740         LD      DE,(SAVEDE)     ;RECALL ADDR PTR
AA79 2A07AA     00750         LD      HL,(SAVEDE)     ;RECALL DATA PTR
AA7C EB         00760         EX      DE,HL           ;PUT EPROM ADDR IN HL
AA7D 7C         00770         LD      A,H             ;EPROM ADDRESS MSB
AA7E CD01AB     00780         CALL    CNVASC          ;GOTO ASCII ROUTINE
AA81 7C         00790         LD      A,H             ;GET ADDR 1ST DIGIT
AA82 320F3E     00800         LD      (VIDEO+527),A   ;ADDR 1ST DIGIT TO SCREEN
AA85 7D         00810         LD      A,L             ;EPROM ADDR  2ND DIGIT
AA86 32103E     00820         LD      (VIDEO+528),A   ;ADDR 2ND DIGIT TO SCREEN
AA89 ED5B07AA   00830         LD      DE,(SAVEDE)     ;RESTORE ADDR PTR
AA8D 2A05AA     00840         LD      HL,(SAVEHL)     ;RESTORE DATA PTR
AA90 EB         00850         EX      DE,HL           ;PUT EPROM LSB ADDR IN HL
AA91 7D         00860         LD      A,L             ;EPROM ADDR LSB
AA92 CD01AB     00870         CALL    CNVASC          ;CONV TO ASCII
AA95 7C         00880         LD      A,H             ;GET ASCII 3RD DIGIT
AA96 32113E     00890         LD      (VIDEO+529),A   ;ADDR 3RD DIGIT TO SCREEN
```

```
AA99 7D        00900          LD      A,L             ;GET 4TH DIGIT
AA9A 32123E    00910          LD      (VIDEO+530),A   ;ADDR 4TH DIGIT TO SCREEN
AA9D ED5B07AA  00920          LD      DE,(SAVEDE)     ;RECALL ADDR PTR
AAA1 2A05AA    00930          LD      HL,(SAVEHL)     ;RECALL DATA PTR
AAA4 3A09AA    00940          LD      A,(SAVEA)       ;RECALL EPROM DATA
AAA7 CD01AB    00950          CALL    CNVASC          ;CONV TO ASCII
AAAA 7C        00960          LD      A,H             ;GET DATA 1ST ASCII CHAR
AAAB 321F3E    00970          LD      (VIDEO+543),A   ;OUT EPROM DATA 1ST DIGIT
AAAE 7D        00980          LD      A,L             ;GET DATA 2ND ASCII CHAR
AAAF 32203E    00990          LD      (VIDEO+544),A   ;OUT EPROM DATA 2ND DIGIT
AAB2 2A05AA    01000          LD      HL,(SAVEHL)     ;RECALL DATA PTR
AAB5 7E        01010          LD      A,(HL)          ;GET INPUT DATA
AAB6 CD01AB    01020          CALL    CNVASC          ;CONV TO ASCII
AAB9 7C        01030          LD      A,H             ;INP DATA 1ST ASCII CHAR
AABA 32333E    01040          LD      (VIDEO+563),A   ;SEND TO SCREEN
AABD 7D        01050          LD      A,L             ;INP DATA 2ND ASCII CHAR
AABE 32343E    01060          LD      (VIDEO+564),A   ;SEND TO SCREEN
AAC1 11873E    01070          LD      DE,VIDEO+647    ;SCREEN POINTER
AAC4 21A1AB    01080          LD      HL,MSG4         ;RETEST MESSAGE
AAC7 012700    01090          LD      BC,39           ;NO OF CHARS
AACA EDB0      01100          LDIR                    ;SEND  MESSAGE TO SCREEN
               01110 ;
               01120 ;---------KEYBOARD SCAN ROUTINE-------------------------
               01130 ;
AACC CD4900    01140 KYBRD    CALL    0049H           ;CALL KYBRD SCAN RTN
AACF FE59      01150          CP      59H             ;IS INPUT CHAR=Y?
AAD1 CADBAA    01160          JP      Z,RESTOR        ;IF SO,RESTORE ALL REGS
AAD4 FE4E      01170          CP      4EH             ;IS CHAR=N?
AAD6 CAF6AA    01180          JP      Z,DOS           ;IF SO, END TEST
AAD9 18F1      01190          JR      KYBRD           ;KEEP SCANNING
               01200 ;-----------------------------------------------------
AADB 3A09AA    01210 RESTOR   LD      A,(SAVEA)       ;RESTORE
AADE ED4B03AA  01220          LD      BC,(SAVEBC)     ;  ALL
AAE2 ED5B07AA  01230          LD      DE,(SAVEDE)     ;       REGISTERS
AAE6 2A05AA    01240          LD      HL,(SAVEHL)     ;                AND:
AAE9 C33BAA    01250          JP      TSTAGN          ;CONTINUE TESTING
               01260 ;
               01270 ;-----------DISPLAY TIMER ROUTINE-----------------------
               01280 ;
AAEC 0100FA    01290 WAIT     LD      BC,64000
AAEF 0B        01300 WAIT1    DEC     BC
AAF0 78        01310          LD      A,B
AAF1 B1        01320          OR      C
AAF2 C2EFAA    01330          JP      NZ,WAIT1
AAF5 C9        01340          RET
               01350 ;-----------------------------------------------------
AAF6 3E9A      01360 DOS      LD      A,9AH           ;CMD WRD FOR
               01370                                  ;A&B PORTS =IN,C=OUT
AAF8 D303      01380          OUT     (03),A          ;SEND IT OUT
AAFA 3EA0      01390          LD      A,0A0H          ;CMD TO TURN OFF ALL
AAFC D302      01400          OUT     (02),A          ;PWR TO EPROM
AAFE C32D40    01410          JP      402DH           ;GO BACK TO DOS
               01420 ;
               01430 ;--------CONVERT TO ASCII ROUTINE-----------------------
               01440 ;
AB01 4F        01450 CNVASC   LD      C,A             ;SAVE HEX DIGITS
AB02 CB3F      01460          SRL     A               ;ALIGN HI DIGMT
AB04 CB3F      01470          SRL     A
AB06 CB3F      01480          SRL     A
AB08 CB3F      01490          SRL     A
AB0A CD16AB    01500          CALL    TEST            ;CNV TO ASCII
AB0D 67        01510          LD      H,A             ;SAVE FOR RETURN
AB0E 79        01520          LD      A,C             ;RESTORE ORIGINAL
AB0F E60F      01530          AND     0FH             ;GET LOW DIGIT
AB11 CD16AB    01540          CALL    TEST            ;CONV TO ASCII
AB14 6F        01550          LD      L,A             ;SAVE FOR RETURN
AB15 C9        01560          RET
AB16 C630      01570 TEST     ADD     A,30H           ;CONVERSION FACTOR
AB18 FE3A      01580          CP      3AH             ;TEST FOR 0-9
AB1A FA1FAB    01590          JP      M,TEST1         ;GO IF 0-9
```

Program continued

```
AB1D C607      01600         ADD     A,7             ;ELSE CORRECT FOR A-F
AB1F C9        01610 TEST1   RET                     ;RETURN
               01620 ;-------------------------------------------------------
AB20 56        01630 MSG1    DEFM    'VERIFYING EPROM'
AB2F 56        01640 MSG2    DEFM    'VERIFICATION COMPLETE....EPROM IS OK IF NO FAILURES DISPLAYED
AB6C 46        01650 MSG3    DEFM    'FAILED ADDRESS:    H.....READS:  .....SHOULD READ:   '
ABA1 44        01660 MSG4    DEFM    'DO YOU WISH TO CONTINUE TESTING ? (Y/N)'
0000           01670         END
00000 TOTAL ERRORS
```

HOME APPLICATIONS

Autocost
Celestial Software
Your Personal Expense Account

Autocost

by Jim Heid

Transportation costs are rising at a frantic pace. The price of gasoline has nearly quadrupled since 1971, and a new sub-compact car costs more than twice as much as it would have 10 years ago. Many consumers have no choice but to finance a new car over a four or five year period. Regular maintenance and sensible driving are vital if the owner expects the car to outlive its payment book.

The Autocost Program

Autocost (see Program Listing) is a Disk BASIC program which maintains gas mileage and maintenance information, displays and, if desired, prints a detailed gas mileage and maintenance cost report. The automobile expense report (see Figure 1) displays the number of miles driven to date, total gallons of gas used, highest miles per gallon (MPG) recorded, lowest MPG recorded, average MPG, number of times that the tank was filled, and the total fuel expense, as well as all maintenance items done, the date each one was done, the cost of each one, and the maintenance cost per mile.

The version of Autocost printed here runs on a 32K, lowercase TRS-80 Model I with one disk drive under Apparat's NEWDOS/80, Version 1.0. Several small modifications make it compatible with NEWDOS 2.1, NEWDOS/80 Version 2, or TRSDOS and with uppercase-only machines. The program has a minimum of multi-statement lines so that its logic and flow are easy to understand and to make it easy to modify.

Running under TRSDOS or NEWDOS 2.1

To run under TRSDOS or NEWDOS 2.1, enter the program as listed, then make the following changes.

1) Delete lines 1040–1070.

2) If the machine has lowercase capability and uses a Radio Shack lowercase driver, change line 1190 to:

```
1190  POKE 16409,0
```

3) If the machine is uppercase only, delete lines 1160–1200.

4) If you wish to keep records for more than one car, EDIT lines 1300 and 1310, and delete the REMs.

5) EDIT line 1550. Delete the CMD"BREAK,Y".

6) Save the altered version of the program to disk. You may wish to save the original version, should you switch operating systems at a future time.

AUTOMOBILE EXPENSE REPORT—07/05/81

For: 81 Toyota Tercel

Miles driven to date	5370.4
Total gallons used	172.993
Highest MPG recorded	38.7618
Lowest MPG recorded	22.4026
Average miles per gallon	30.5822
No. of times tank filled	18
Total fuel expense	$230.24

Maintenance Report:

Maintenance Item	Date Done	Cost
Oil change & filter	06/14/81	$ 14.52
New wiper blades	06/14/81	$ 4.35
Oil change & filter	07/01/81	$ 15.00

Total Maintenance Cost:
$ 33.87
Maintenance Cost per mile:
$ 0.01

Figure 1. *Automobile expense report*

Running under NEWDOS/80, Version 2.0

Make the following changes to run the program under Version 2.0.

1) If the machine has lowercase capability, EDIT line 1190 to read:

```
1190  CMD"LC,Y"
```

2) If it is an uppercase only machine, delete lines 1160–1200.

3) If you wish to keep records for more than one car, EDIT lines 1300 and 1310 and delete the REMs.

4) Save the altered version of the program. You may wish to have the original version, should you change operating systems at a future time.

Running under NEWDOS/80, Version 1

1) If you machine is a lowercase, 32K machine, EDIT line 1190 to read:

```
1190  POKE(&HBFB2,&H20)
```

2) If it is a lowercase, 48K machine, EDIT line 1190 to read:

```
1190  POKE(&HFFB2,&H20)
```

3) If you have an uppercase-only machine, DELETE lines 1160–1200.

Using the Program

When you run Autocost, enter the date in DD/MM/YY format. Be sure to enter the date correctly, because the program does not check for impossible day or month numbers. (Example: If it is August 12, 1981, type 08/12/81 and press ENTER.) At this point, the main menu appears (Figure 2). The menu is the branch point to each different area of the program. Your response determines what the program does. You see a list of numbered options, followed by the word *Choice?* and a white square. The white square means that you must respond by striking only one key. You do not need to press ENTER when the white square appears.

Starting Out

The first time you run the program, you need to tell the computer a few things about your car. Do this by selecting option 4, *Start New Data Set*. The computer asks you several questions. The first asks you to type in the year and name of the car (for example: 81 Ford, 73 Dodge). The next question asks for the car's current mileage. Enter the mileage with no commas between any digits.

Next, the computer asks you if the car is new. If it is not new, you are warned that any maintenance cost data will not be accurate since you've already had work done on the car that was not recorded. Finally, it asks you whether the data you entered before is correct. If it is, strike Y. The data you have entered is saved on disk. If you have entered something incorrectly, strike N and reenter the data. If you've changed your mind about the whole thing, strike A (for abort) to return to the main menu.

```
* AUTOCOST 1.0 *

(1) Enter Fuel Data
(2) Enter Maintenance Data
(3) Print Report(s)
(4) Start New Data Set
(5) End Program Run

Choice?
```

Figure 2. *The main menu*

Entering Fuel Data

To maintain gas mileage figures, first fill your car's gas tank. Wait until your tank is about 3/4 empty before filling it again. Write down the current mileage, the number of gallons that just went into the tank, and the price per gallon.

Now select option 1 of the main menu, *Enter Fuel Data.* After you strike a 1, the screen will say, *DISK CONTAINING DATA FILES MUST BE IN DRIVE #0. Strike ENTER to begin, any other key to exit.* Make sure that the same disk you used when you started the new data set is in drive 0, then strike the ENTER key. The computer reads your car's data, shows the mileage at your last fill-up, and asks the following questions:

- MILEAGE READING AT THIS FILL-UP ? (Enter the mileage reading that you just wrote down, without commas. Example: 3,453.8 miles would be entered as 3453.8.)
- GALLONS PURCHASED AT THIS FILL-UP ? (Enter the number of gallons you just bought.)
- PRICE PER GALLON ? (Enter the price per gallon that you just paid, without dollar signs. Example: $1.50 would be entered as 1.50.)

The computer now asks, *IS ABOVE DATA CORRECT (Y/N/A)*? Check what you entered carefully. If there is an error, strike N and reenter the data. If everything is correct, strike Y. If you're frustrated and need a drink, strike A (for abort) to return to the main menu with no changes made.

If everything was correct, and you struck Y, the screen now shows:

```
(Type of car) got (20) miles per gallon,
using (5) gallons. Miles driven: (100)
Cost per mile: ($0.04)
Total Fuel Cost to Date: ($10.00)
```

The figures in parentheses are different for each car. At the same time, the disk drive activates, saving the newly updated data files. When you are finished looking at the display, strike any key to return to the main menu. The above steps are easier to do than they are to describe. Just remember to record your mileage, gallons bought, and price per gallon each time you fill the tank.

Recording Maintenance Data

Entering maintenance data is much like entering fuel data. Begin by selecting option 2 from the main menu, *Enter Maintenance Data.* Make sure your car disk is in drive 0, then press ENTER. The computer reads your car's data files and responds by asking:

- ENTER DATE OF MAINTENANCE (DD/MM/YY)? (Enter the date that the work was done. Example: If the work was done on June 20, 1981, enter 06/20/81.)
- ENTER SUMMARY OF WORK DONE (35 CHAR. MAXIMUM) ? (Type a short description of what was done. Your description must be under 35 characters—if it is longer, the computer asks you to enter it again. Examples of descriptions include oil change and filter, new tires, flush coolant system.)
- ENTER COST OF MAINTENANCE ? (Enter the cost of the

maintenance, using no commas or dollar signs. Example: If the work cost $45.36, enter 45.36.)
The computer asks if the data is correct. If what you entered is incorrect, type N and reenter the data. If the information is correct, type Y. If you've decided to sell the car and buy a horse, type A (for abort), and you return to the main menu with no changes made. If the data was correct and you reply with Y, the computer updates the maintenance data file and returns to the main menu.

Viewing Reports

One of the great advantages of record keeping by computer is that detailed, concise reports can be displayed or printed at any time. Option 3, *Print Reports*, allows this. After you type a 3, a smaller menu appears on the screen (see Figure 3). This is a sub-menu, the branch point for the report generating modules of Autocost.

The first selection of the sub-menu is 1, *Print Expense Report.* This module reads the data files, computes highest, lowest, and average gas mileage, displays cost per mile, fuel cost to date, maintenance work, dates and cost of maintenance, and maintenance cost per mile (shown in Figure 1). Note: The report will display meaningless figures if you call it up after making your first fuel consumption entry. There must be more than one entry on the disk to display an accurate report.

To view these reports, type a 1 from the sub-menu. The first report (fuel consumption) is displayed. There is a line at the bottom of the screen saying *END OF GAS REPORT—STRIKE <P> TO PRINT, <N> FOR NEXT REPORT.* If you want a printout of the fuel report, strike P. The program warns you if your printer is not ready. If you do not want a printout of the fuel report, press N (for next report). The program then displays the maintenance data for your car: maintenance item, date done, cost, total maintenance cost, and maintenance cost per mile. If there are more than 11 maintenance items, the display pauses until you strike N. At the end of the report, you may obtain a printout by striking P. Again, you are warned if your printer is not ready. To return to the main menu, strike X.

```
** DISK CONTAINING DATA FILES MUST BE IN DRIVE #0
        REPORT PRINTING MODULE
    1. Print Expense Report
    2. Print Gas Mileage Graph
    3. Return to Menu
            Choice?
```

Figure 3. *The sub-menu*

Option 2 of the sub-menu, *Print Gas Mileage Graph*, displays your car's gas mileage records in a bar graph form. To view the graph, strike a 2 from the sub-menu. The data is read from the disk, and the graph is displayed. To the right of each bar is the exact gas mileage figure. If there are more than 15 figures to display, the display pauses until you strike an N (for next screen). When all of the entries have been shown, you may either see the entire graph again by striking A, or you may return to the main menu by striking X.

Option 3 of the sub-menu returns you to the main menu. Use this if you selected *(3) Print Report(s)* from the main menu by mistake.

Ending the Run

Select main menu option 5 only when you wish to end the entire program run. For NEWDOS/80 users, option 5 is the only way to regain control of the computer since this program automatically disables the BREAK key. For users of other operating systems, the use of the BREAK key to exit the program is discouraged.

To end the program run, strike a 5. Under the menu selections, you see *VERIFY END ? (Y/N)*. If you wish to end the run, strike Y. The computer's string space is reallocated to 50, the screen clears, NEWDOS/80 reenables the BREAK key, and the computer returns to the command mode. If you do not wish to end the run, strike an N, and the main menu reappears.

Inside Autocost

This information tells how the program operates and how to modify or customize it. This section is also of value to the novice programmer since it contains many techniques useful in any program.

Autocost is written in Disk BASIC, using a minimum of multi-statement lines. All line numbers contain four digits, and all statements are indented, resulting in an easy-to-read program listing. It uses no variable definition statements (DEFSTR, DEFINT). They tend to make a listing harder to understand, since you must either remember which variables are what type, or you must constantly refer to the definition statements. The program, as written, fits and operates well within 32K of memory, but could be compressed considerably by combining statement lines and deleting REMarks. Table 1 lists the variables Autocost uses.

Lines 1040–1070 disable the BREAK key in NEWDOS/80 to prevent an awkward exit that could lead to a loss of data. The routine in lines 1080–1150 eliminates the need to set the date under DOS. It first checks to see if you have already entered the date. This is done by PEEKing at location 16454 (decimal). If the value is greater than zero, the routine assumes that the date has been entered and jumps to line 1160.

If PEEK(16454) is zero, the computer asks you to input the date into string variable D$. The string is then peeled apart, with the left two

Name	Type	Use
A	Single-precision	Total of MPG values
AG	Single-precision	Average MPG
CM	Single-precision	Current mileage
CPM	Single-precision	Cost per mile
D	String	Current date
DD	String array	Date done (maintenance)
F1	String	Filespec #1 (GASFILE/DAT)
F2	String	Filespec #2 (MAINTFLE/DAT)
GU	Single-precision	Gallons used
K	String	INKEY$ keystroke string
MC	Single-precision array	Maintenance item cost
MD	Single-precision	Miles driven
MI	String array	Maintenance item
MPG	Single-precision array	Miles per gallon
N1	Single-precision	Number of fill-ups
N2	Single-precision	Number of maintenance items
NM	Single-precision	New mileage (for MPG data)
P	String	Used in report printing
PG	Single-precision	Price per gallon
T	Single-precision	Temporary storage (sort)
TC	String	Type of car
TFC	Single-precision	Total fuel cost (to date)
TGU	Single-precision	Total gallons used
TM	Single-precision	Total maintenance cost
X	Single-precision	General work variable
X1	Single-precision	General work variable
X2	Single-precision	General work variable
Z	Single-precision	General work variable

Table 1. *Variable list*

characters (the day) POKEd into location 16454, the right two characters (the year) POKEd into 16452, and the middle two characters (the month), going to location 16453. At these three locations, the date is stored and accessed by the TIME$ function.

Lines 1160–1200 assure that the NEWDOS/80 lowercase driver is set to capitals and lowercase. Since all string comparisons in Autocost require lowercase input, it is vital that the lowercase driver be enabled.

Lines 1210–1280 clear 1000 bytes of string space, dimension the four arrays used to 55, and assign the two filenames you will use for data files. If you have more than 32K of memory you may want to increase the DIMensioning of arrays MI$(X), DD$(X), MC(X), and MPG(X). As written, the dimension of 55 will allow the data from a total of 55 fill-ups and maintenance entries. Lines 1300 and 1310 are for users who want to keep records for more than one car. If you delete the REMs, the program prompts you for two filenames.

The code in lines 1330–1470 displays the main menu and branches to an INKEY$ subroutine that waits for keyboard input. Upon receiving a keystroke, the code tests the numeric value of K$. If it is 0 or greater than 5, the menu appears (line 1440). If it is between 0 and 5, control branches to the appropriate routine (line 1460).

Lines 1480–1560 make up the verify end routine. This routine is called if you strike a 5 from the main menu. It asks you to verify the end by replying Y or N. If the reply is neither a Y nor an N, the program loops back to line 1500 *(VERIFY END ? Y/N)* until you give a legal response. If the response is an N, control proceeds back to the main menu. If the response is a Y (end), the screen clears, a token CLOSE is given, string space is CLEARed to 50 bytes, the BREAK key is reenabled, and the computer returns to the command mode (line 1550).

Lines 1570–2260 make up the fuel consumption module. Lines 1590–1670 allow you to return to the main menu if you accidentally struck 1. Since you must press ENTER to begin, entry into this module is a two-step process, eliminating the chance of getting there by mistake.

Lines 1700–1820 input the data needed to compute gas mileage. Lines 1830–1880 provide yet another exit if you have entered the data incorrectly or want to return to the menu. The message *IS ABOVE DATA CORRECT (Y/N/A) ?* is displayed. If you respond N, the program returns to line 1700, requiring reentry of the data. If you respond A, the program exits this module and returns to the main menu with no changes made in the data files. If you respond Y, execution continues. Responding with any other key causes a loop back to 1830.

Lines 1890–2120 compute the number of miles driven (MD), the miles per gallon (MPG), the cost per mile (CPM), the total fuel cost (TFC), and the total number of gallons used (TGU). The current mileage (CM) is updated, and the MPG subscript (variable N1) is increased by one. Lines 2130–2260 display the car's current gas mileage figures and write the new data files to disk.

The maintenance data module (lines 2270–2630) operates much the same as the fuel consumption module. Lines 2290–2360 provide an exit if you accidentally struck a 2. Line 2370 branches control to the file read subroutine which reads previously entered data.

Lines 2390–2520 input the necessary data about the current maintenance item. Line 2420 asks you to enter the date of maintenance into string array DD$(X). Note that the array's subscript (N2) does not increase by one at this point. This is to allow for a possible exit, should you decide to abort to the main menu.

Line 2430 requests a 35-character or fewer description of maintenance work done. This data is input in line 2440 to string array MI$(X). The length of the description (in MI$(X)) is tested at line 2460. If the length is legal

(fewer than 36 characters), execution jumps to line 2510. If the length is over 35 characters, an error message appears (lines 2470–2480), and execution loops back to 2430.

Line 2510 inputs the cost of the maintenance item into array MC(X). Again note that the subscript variable (N2) does not increment by one, but has one added to it within the subscript. Lines 2530–2600 ask if the input data is correct. As in the fuel consumption module, you have a chance to abort to the main menu with no changes in the data files.

The program reaches line 2620 only if you respond Y to *IS ABOVE DATA CORRECT (Y/N/A)* ? This line adds one to the subscript variable, updates the maintenance cost to date (variable TM), writes the updated data files, and returns control to the main menu.

Lines 2640–2780 display the sub-menu that is the branch point to the report printing modules. Line 2760 tests the numerical value of K$ (the string variable used in the INKEY$ subroutine), and branches to the appropriate routine. The logic in this section of code is identical to that of the main menu.

Lines 2790–3420 generate the fuel consumption report. After the fuel data is read from disk (line 2810), lines 2850–2920 sort the data in the miles per gallon array, MPG(X), from lowest to highest, using a basic Shell sort.

Lines 2940–2970 compute the average miles per gallon by totaling all the MPG entries (lines 2940–2960) and dividing the total by the number of entries (line 2970).

Lines 2990–3120 display the gas mileage report on the screen. Lines 2990 and 3010 use the STRING$ function to print a row of hyphens on either side of the report title. Line 3010 uses the LEFT$ function to obtain the date.

Lines 3040–3110 print the report information. The item descriptor (such as miles driven to date, total gallons used) is printed, followed by a row of periods, then the actual value. The periods, or leaders, serve to lead the eye across the line, resulting in an easy-to-read report.

The maximum line width of the report is 55 characters. For a pleasing visual appearance, the left and right sides of the report should be flush with the 55-character measure. To accomplish this you must know how many digits will be in the data, how long each item descriptor is, and how many leaders to put between the end of the item descriptor and the beginning of the figures.

Make seven the maximum number of digits in the data. You must also include a period for decimal values. This sets our figure length at eight. If you subtract eight from the total measure (55), you get 47. Thus you have to print the item descriptor and enough leaders to fill 47 spaces. This brings the leaders to the start of the figures and also prints the figures flush right on the 55-character measure.

The only problem now is computing how many leaders to print on each

line. The program accomplishes this in the following steps:
1) The program assigns the item descriptor to string variable P$.
2) The program finds the length of P$ (the item descriptor) and subtracts it from 47 (total measure minus eight for figures) then assigns the difference to variable X.
3) It changes P$ to equal the original item descriptor plus X number of periods (X being the difference between the length of the item descriptor and the total width of 55 characters).
4) Finally it prints the new P$, then the figures.
While this may sound complex, the code needed to accomplish it is quite simple. To help you understand it, here is an example:

```
10 REM  Demonstrator of "leader" routine
20 REM  Maximum line width = 55
30 REM
40 P$ = "Here is item descriptor"
50 X = 47 - LEN(P$)
60 P$ = P$ + STRING$(X,".")
70 PRINT P$; "12345.67"
80 PRINT : END
```

In the Autocost program, a subroutine (lines 5130–5170) adds the leaders to P$. Other than that, the logic is identical to the above demonstration program.

Lines 3130–3170 display a message at the bottom of the screen, giving you the option to line print the gas mileage report or simply move on to display the maintenance data report. If you choose not to print the gas mileage report, execution jumps to line 3430.

Lines 3190–3410 print the MPG report on the line printer. Lines 3210–3240 check the status of the line printer. Line 3210 PEEKs the printer device control block, address 14312 (decimal). If the value found there is less than 127, the printer is off-line or not connected. The screen displays the message *PRINTER NOT READY*! and the program loops back to line 3130.

If the value of 14312 is 127 or greater, the screen displays <<*PRINTING*>>, and the report is printed, using code identical to that which printed the report on the screen. After the report is printed, the program automatically displays the maintenance report.

Program lines 3430–3950 generate the maintenance data report. Lines 3450–3690 display the report on the screen. Line 3540 sends execution to a screen pause subroutine if the screen is full. Press N to see the next screen; press X to exit to the main menu.

Lines 3700–3950 contain the code needed to line print the maintenance report. Lines 3720–3750 test the status of the printer in the manner described above. Lines 3770–3950 print the report, again using the same code

as the display. After the report is printed, the program returns to the main menu.

Lines 3970–4170 display a bar graph of the car's gas mileage data. Line 3990 sends execution to the file read subroutine. After the data is read, a FOR-NEXT loop is initialized (line 4020). Line 4050 uses the STRING$ function to display a bar of graphics characters (character 140) to the length of the gas mileage figure that X points to. That mileage figure is then printed to the immediate right of the bar. For example, assume that X (the work variable in the FOR-NEXT loop) equals 3. The current MPG entry in the array MPG(3) equals 24.323 (miles per gallon). Line 4050 uses the integer portion of 24.323 to display a bar of 24 white blocks. The current MPG value appears immediately to the right of that bar. In this case, the value is 24.323.

Line 4030 sends execution to the screen pause subroutine if the screen is full of lines. When the entire graph has been displayed, you may either see it from the beginning by striking A or return to the main menu by striking X.

Lines 4190–4490 make up the data initialization module. The program inputs the necessary data in lines 4210–4260. Lines 4260–4380 print a warning about data accuracy, as described earlier. Lines 4400–4490 ask if the newly entered data is correct. If the response is Y (line 4450), the new data files are written, and the program returns to the main menu. If the response is A (line 4460), the program aborts to the main menu without making changes in the data files. If the response is N, execution loops back to line 4220 where you must reenter the data.

The rest of the program consists of subroutines. Lines 4510–4750 open and write the two sequential data files to disk. Sequential files are used for their simplicity, and because random access files offer no advantage to this program's execution. Lines 4550–4560 open the two files under the names contained in variables F1$ and F2$. Line 4580 writes the data to the beginning of file 1 (GASFILE/DAT, unless you change it). The current mileage (CM), total gallons used (TGU), number of fill-ups (N1), cost per mile (CPM), and total fuel cost (TFC) are then written.

Line 4600 writes the number of maintenance items (N2) to the beginning of file 2 (MAINTFLE/DAT, unless changed). Lines 4620–4650 use a FOR-NEXT loop to write each maintenance item description (MI$(X)), each date done (DD$(X)), and the cost of each maintenance item (MC(X)).

Lines 4670–4690 use a FOR-NEXT loop to write each MPG value (MPG(X)) to file 1. Line 4710 writes the miles driven (MD) and the total maintenance cost (TM) to file 2. Line 4730 closes both files, and line 4740 returns execution to the main program. Lines 4760–4960 open and input the data from the files in the same order and to the same variables as it was written.

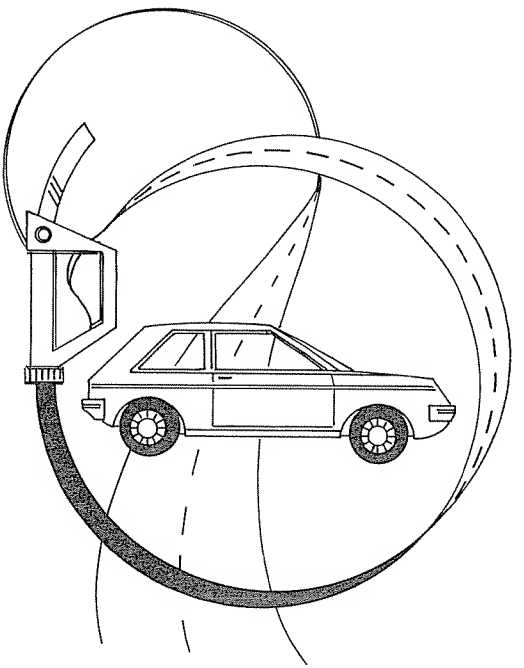
Lines 4980–5070 make up the screen pause subroutine. *STRIKE <N> FOR NEXT SCREEN, <X> TO EXIT*, appears at the bottom of the screen.

Execution then jumps to the INKEY$ subroutine until you strike a key. If the keystroke is N, the screen clears, and control returns to the main program (line 5050). If the keystroke is anything other than X (line 5060), execution loops back to display the message (line 5000). If the keystroke was X, execution is turned over to the main menu.

Lines 5090–5120 make up a conventional INKEY$ subroutine. Variable K$ is assigned to INKEY$. If you have not struck a key (K$ equals null or K$ = ""), execution loops back to the INKEY$ statement. If you did strike a key, execution returns to the main program, where the contents of K$ can be tested for certain responses. Lines 5130–5170 in the report generating module determine how many periods, or leaders, to display.

Summing It Up

The Autocost program is a good example of how simple programming techniques can work together to make a useful piece of software. The style of the program—its single-statement, indented lines and remarks—make for clearer understanding of its operation, and make modifications or customizing much simpler.



Program Listing

Encyclopedia Loader

```
1000 REM
1010 REM                "AUTOCOST"  - VERSION 1.1
1020 REM                COPYRIGHT 1981 BY JIM HEID
1030 '
1040 REM       DISABLE BREAK (DELETE IF ON TRSDOS OR NEWDOS+)
1050 '
1060     CMD"BREAK,N"
1070 '
1080 REM         INPUT DATE AND POKE INTO TIME$
1090 '
1100    IF PEEK(16454) >0 GOTO 1160
1110     CLS : PRINT@384,;:INPUT "ENTER DATE (DD/MM/YY)"; D$
1120      POKE 16454, VAL(LEFT$(D$,2))
1130      POKE 16452, VAL(RIGHT$(D$,2))
1140      POKE 16453, VAL(MID$(D$,4,2))
1150 '
1160 REM       TOGGLE U/L TO LOWER CASE (DELETE IF U.C. ONLY)
1170 REM       (THIS ADDRESS FOR NEWDOS/80 VER. 1, 32K)
1180 '
1190     POKE (&HBFB2),(&H20)
1200 '
1210 REM       ****  INITIALIZE & ASSIGN FILE NAMES
1220 REM         (IF MANUAL INPUT OF FILENAMES IS DESIRED,
1230 REM         DELETE LINE 1280, THEN DELETE THE "REMs"
1240 REM         FROM LINES 1300 and 1310)
1250 '
1260      CLEAR 1000
1270      DIM MI$(55), DD$(55), MC(55), MPG(55)
1280      F1$="GASFILE/DAT" : F2$="MAINTFLE/DAT"
1290 '
1300 REM   CLS:INPUT"ENTER FUEL DATA FILESPEC"; F1$
1310 REM   INPUT"ENTER MAINTENANCE DATA FILESPEC"; F2$
1320 '
1330 REM                MENU & BRANCH
1340 '
1350    CLS : PRINT@280, "* AUTOCOST 1.1 *" : PRINT
1360    PRINT@404, "(1)   Enter Fuel Data"
1370    PRINT TAB(20) "(2)   Enter Maintenance Data"
1380    PRINT TAB(20) "(3)   Print Report(s)
1390    PRINT TAB(20) "(4)   Start New Data Set
1400    PRINT TAB(20) "(5)   End Program Run
1410 '
1420    PRINT@768+25, "Choice? " ;CHR$(143)
1430     GOSUB 5090
1440      K = VAL(K$) : IF K=0 OR K >5 THEN 1330
1450 '
1460     ON K GOTO 1570, 2270, 2640, 4190, 1480
1470 '
1480 REM             VERIFY END?
1490 '
1500    PRINT@768+22, "VERIFY END? (Y/N) "; CHR$(143)
1510     GOSUB 5090
1520     IF K$="n" THEN 1330
1530     IF K$<>"y" THEN 1500
1540 '
1550   CLS : CLOSE : CLEAR 50 : CMD"BREAK,Y" : END
1560 '
1570 REM                ENTER FUEL DATA
1580 '
1590    CLS : PRINT@275, "Fuel Consumption Module
1600     PRINT@391, "DISK WITH PROPER DATA FILES MUST BE IN ";
1610     PRINT "DRIVE #0."
1620     PRINT@519, "Strike <ENTER> to begin, any other key ";
1630     PRINT "to exit ";
1640     PRINT CHR$(143)
1650      GOSUB 5090
```

Program continued

```
1660       IF ASC(K$) <> 13 GOTO 1330
1670 '
1680    GOSUB 4760                        'FILE READ
1690 '
1700    CLS : PRINT@256, "Gas Mileage Data Module"
1710     PRINT@384, "Mileage reading at last fill-up: "; CM
1720     PRINT : INPUT "Mileage reading at this fill-up" ; NM
1730       IF NM > CM THEN GOTO 1770
1740       PRINT"*** ERROR! *** MILEAGE READING MUST BE"
1750       PRINT"HIGHER THAN READING AT LAST FILL-UP!"
1760       GOTO 1720
1770     INPUT "Gallons purchased at this fill-up" ; GU
1780       IF GU > 1 THEN GOTO 1810
1790       PRINT"*** ERROR! *** RE-ENTER NUMBER OF GALLONS!"
1800       PRINT : GOTO 1770
1810     INPUT "Price per gallon"; PG
1820 '
1830    PRINT@832, "IS ABOVE DATA CORRECT (Y/N/A)? ";CHR$(143)
1840     GOSUB 5090
1850      IF K$="n" THEN 1700
1860      IF K$="a" THEN 1330
1870      IF K$<>"y" THEN 1830
1880 '
1890 REM                Following module calculates miles
1900 REM                per gallon using this process:
1910 REM
1920 REM         MILES DRIVEN = NEW MILAGE - MILEAGE @ LAST FILL
1930 REM         MPG = MILES DRIVEN/GALLONS USED
1940 REM         COST/MILE = PRICE PER GALLON/MILES DRIVEN
1950 REM         TOTAL FUEL COST (TFC) = TFC + (PG * GU)
1960 REM         TOTAL GALLONS USED (TGU) = TGU + GU
1970 '
1980    MD = NM - CM
1990     MPG = MD / GU
2000      CPM = (PG * GU) / MD
2010       TFC = TFC + (PG * GU)
2020        TGU = TGU + GU
2030 '
2040 REM                UPDATE CURRENT MILEAGE TO NEW VALUE
2050 REM                (CM = NM)
2060 REM                INCREASE MPG SUBSCRIPT BY 1 AND ASSIGN
2070 REM                NEW MPG VALUE (N=N+1 : MPG(N)=MPG
2080 '
2090    CM = NM
2100     N1 = N1 + 1  :  MPG(N1) = MPG
2110 '
2120 '
2130 REM               PRINT MPG FIGURES FOR THIS RUN
2140 REM               AND SAVE DATA TO DISK.
2150 '
2160   CLS : PRINT@320, TC$; " got" ; MPG ; "miles per gallon,
2170    PRINT"using"; GU ; "gallons. Miles driven: "; MD
2180    PRINT"Cost per mile: ";: PRINT USING "$##.##"; CPM
2190   PRINT"Total Fuel Cost-to-Date: ";
2200     PRINT USING "$####.##"; TFC
2210 '
2220   GOSUB 4510                        'WRITE NEW FILES
2230 '
2240   PRINT@640, "<< COMPLETED - HIT ANY KEY FOR MENU >>"
2250    GOSUB 5090 : GOTO 1330
2260 '
2270 REM                ENTER MAINTENANCE DATA
2280 '
2290    CLS : PRINT@273, "Maintenance Data-Entry Module"
2300     PRINT@391, "DISK WITH PROPER DATA FILES MUST BE IN ";
2310     PRINT "DRIVE #0."
2320     PRINT@519, "Strike <ENTER> to begin, any other key ";
2330     PRINT "to exit "; CHR$(143)
2340      GOSUB 5090                              'get keystroke
2350       IF ASC(K$) <> 13 GOTO 1330
```

```
2360 '
2370     GOSUB 4760                                    'file read
2380 '
2390    CLS : PRINT@256, "Maintenance Data Module"
2400     PRINT
2410 '
2420    INPUT"Enter date of maintenance (DD/MM/YY) "; DD$(N2+1)
2430    PRINT "Enter summary of work done (35 char. maximum):"
2440    PRINT : LINEINPUT MI$(N2+1)
2450 '
2460    IF LEN(MI$(N2+1)) <36 GOTO 2510
2470     PRINT:PRINT "** ERROR! DESCRIPTION MUST BE UNDER 35 ";
2480      PRINT "CHARACTERS! ** " : PRINT
2490       GOTO 2430
2500 '
2510    INPUT "Enter cost of maintenance" ; MC(N2+1)
2520 '
2530    PRINT:PRINT "IS ABOVE DATA CORRECT? (Y/N/A) ";CHR$(143)
2540     GOSUB 5090                                   'Get keystroke
2550      IF K$="y" THEN GOTO 2620                    'SAVE NEW FILE
2560      IF K$="a" THEN GOTO 1330                    'abort
2570      IF K$<>"n" THEN GOSUB 5090                  'bad entry
2580 '
2590    CLS : PRINT@128, "RE-ENTER THE DATA:"
2600     GOTO 2400

2610 '
2620    N2 = N2+1 : TM = TM + MC(N2) : GOSUB 4510 : GOTO 1330
2630 '
2640    REM              PRINT REPORTS
2650 '
2660    CLS : PRINT@128+8, "** DISK CONTAINING DATA FILES ";
2670    PRINT "MUST BE IN DRIVE #0"
2680     PRINT@384+20,"REPORT PRINTING MODULE"
2690     PRINT
2700     PRINT TAB(20) "1. Print Expense Report"
2710     PRINT TAB(20) "2. Print Gas Mileage Graph
2720     PRINT TAB(20) "3. Return to Menu
2730     PRINT : PRINT TAB(20) "Choice? "; CHR$(143)
2740 '
2750     GOSUB 5090                          'GET KEYSTROKE
2760      K=VAL(K$) : IF K=0 OR K>3 THEN 2750
2770       ON K GOTO 2790, 3960, 1330
2780 '
2790 REM                 PRINT EXPENSE REPORT
2800 '
2810     GOSUB 4760                          'READ FILE
2820 '
2830 REM        COMPUTE HIGHEST, LOWEST, AVERAGE MPG'S
2840 '
2850    FOR X1 = 1 TO N1 - 1
2860     FOR X2 = X1+1 TO N1
2870        IF MPG(X1) <= MPG(X2) THEN 2910
2880        T = MPG(X1)
2890        MPG(X1)=MPG(X2)
2900        MPG(X2)=T
2910      NEXT X2
2920     NEXT X1
2930 '
2940     FOR X1 = 1 TO N1
2950       A = A + MPG(X1)
2960     NEXT X1
2970       AG = A/N1
2980 '
2990     CLS : PRINT STRING$(55,"-")
3000     PRINT TAB(9) "AUTOMOBILE EXPENSE REPORT - ";
3010     PRINT LEFT$(TIME$,8) : PRINT STRING$(55,"-")
3020     PRINT "For: "; TC$
3030     PRINT
3040     P$="Miles driven to date" : GOSUB 5130 : PRINT P$ ;CM
```

Program continued

```
3050    P$="Total gallons used" : GOSUB 5130 : PRINT P$; TGU
3060    P$="  Highest mpg recorded":GOSUB 5130:PRINT P$;MPG(N1)
3070    P$="  Lowest mpg recorded": GOSUB 5130:PRINT P$;MPG(1)
3080    P$="  Average miles per gallon":GOSUB 5130:PRINT P$;AG
3090    P$="No. of times tank filled":GOSUB 5130:PRINT P$;N1
3100    P$="Total fuel expense":GOSUB 5130:PRINT P$;:
3110     PRINT USING "$####.##"; TFC
3120 '
3130    PRINT@960,"END OF GAS REPORT - STRIKE <P> TO PRINT, ";
3140      PRINT "<N> FOR NEXT REPORT ";
3150    PRINT CHR$(143); : GOSUB 5090
3160     IF K$="n" THEN 3450
3170     IF K$<>"p" THEN 3130
3180 '
3190 REM        LINE-PRINT GAS MILEAGE REPORT
3200 '
3210    IF PEEK(14312)<127 THEN 3260            'PRINTER READY
3220    PRINT@960, STRING$(63," ");
3230    PRINT@960, "* PRINTER NOT READY! *";
3240    FOR Z=1 TO 1000 : NEXT : GOTO 3130
3250 '
3260    CLS : PRINT@474, "<< PRINTING >>" ;
3270 '
3280    LPRINT STRING$(55,"-")
3290    LPRINT TAB(9) "AUTOMOBILE EXPENSE REPORT - ";
3300    LPRINT LEFT$(TIME$,8) : LPRINT STRING$(55,"-")
3310    LPRINT "For: "; TC$
3320    LPRINT CHR$(138)
3330    P$="Miles driven to date" : GOSUB 5130 : LPRINT P$ ;CM
3340    P$="Total gallons used" : GOSUB 5130 : LPRINT P$; TGU
3350   P$="  Highest mpg recorded":GOSUB 5130:LPRINT P$;MPG(N1)
3360    P$="  Lowest mpg recorded": GOSUB 5130:LPRINT P$;MPG(1)
3370    P$="  Average miles per gallon":GOSUB 5130:LPRINT P$;AG
3380    P$="No. of times tank filled":GOSUB 5130:LPRINT P$;N1
3390    P$="Total fuel expense":GOSUB 5130:LPRINT P$;:
3400     LPRINT USING "$####.##"; TFC
3410   LPRINT STRING$(3,138)
3420 '
3430 REM              DISPLAY MAINTENANCE REPORT
3440 '
3450    CLS
3460    PRINT "Maintenance Report:" : PRINT
3470    PRINT "Maintenance Item" TAB(32) "Date Done" ;
3480    PRINT TAB(49) "Cost"
3490    PRINT STRING$(55,"-")
3500 '
3510    FOR X = 1 TO N2
3520      PRINT MI$(X) TAB(32) DD$(X) TAB(46);
3530      PRINT USING "$###.##"; MC(X)
3540      IF X=11 OR X=22 OR X=33 OR X=44 GOSUB 4980
3550      NEXT X
3560 '
3570    PRINT STRING$(55,"-")
3580    PRINT "Total Maintenance Cost: "
3590    PRINT USING"$####.##"; TMC
3600    PRINT "Maintenance Cost-per-mile:"
3610     PRINT USING "$##.##"; TMC/CM
3620 '
3630    PRINT@960,"END OF REPORT - STRIKE <P> TO PRINT, ";
3640    PRINT "<X> TO EXIT ";
3650    PRINT CHR$(143); : GOSUB 5090
3660 '
3670    IF K$="x" THEN 1330
3680    IF K$<>"p" THEN 3630
3690 '
3700 REM        LINE-PRINT MAINTENANCE REPORT
3710 '
3720    IF PEEK(14312)<127 THEN 3760            'PRINTER READY
3730    PRINT@960, STRING$(63," ");
3740    PRINT@960, "* PRINTER NOT READY! *";
```

```
3750    FOR Z=1 TO 1000 : NEXT : GOTO 3630
3760 '
3770    CLS : PRINT@474, "<< PRINTING >>" ;
3780    LPRINT "Maintenance Report:" : LPRINT CHR$(138)
3790    LPRINT "Maintenance Item" TAB(32) "Date Done" ;
3800    LPRINT TAB(49) "Cost"
3810    LPRINT STRING$(55,"-")
3820 '
3830    FOR X = 1 TO N2
3840      LPRINT MI$(X) TAB(32) DD$(X) TAB(46);
3850      LPRINT USING "$###.##"; MC(X)
3860      NEXT X
3870 '
3880    LPRINT STRING$(55,"-")
3890    LPRINT "Total Maintenance Cost: "
3900    LPRINT USING"$####.##"; TMC
3910    LPRINT "Maintenance Cost-per-mile:"
3920    LPRINT USING "$##.##"; TMC/CM
3930 '
3940    LPRINT STRING$(5,138)
3950    GOTO 1330                                'MENU
3960 '
3970 REM                PRINT MPG GRAPH
3980 '
3990    GOSUB 4760                         'READ DISK
4000 '
4010    CLS
4020     FOR X = 1 TO N1
4030     IF X = 15 OR X = 30 THEN GOSUB 4980
4040 '
4050     PRINT STRING$(MPG(X),140) ;
4060     PRINT USING " ##.####"; MPG(X)
4070 '
4080     NEXT X
4090 '
4100     PRINT@960,"END OF DATA - STRIKE <A> TO SEE AGAIN, ";
4110     PRINT "<X> TO EXIT ";
4120     PRINT CHR$(143) ;
4130     GOSUB 5090                              'GET KEYSTROKE
4140 '
4150     IF K$="a" THEN 4010
4160     IF K$<>"x" THEN 4130
4170     GOTO 1330                                'MAIN MENU
4180 '
4190 REM                START NEW DATA SET
4200 '
4210     CLS : PRINT@192, "NEW DATA SET - INITIALIZATION DATA:"
4220     PRINT
4230      PRINT "Type of car (";CHR$(34);"81 Ford";CHR$(34);")";
4240       INPUT TC$ : IF TC$="x" GOTO 1330
4250     INPUT "Current mileage (NO COMMAS)"; CM
4260     PRINT "Is the "; TC$ ; " new? (Y or N) "; CHR$(143)
4270      GOSUB 5090
4280       IF K$="y" THEN GOTO 4400
4290       IF K$<>"n" THEN GOTO 4270
4300 '
4310 REM                CAR ISN'T NEW. PRINT REMINDER.
4320 '
4330     PRINT "*** NOTE! ***
4340     PRINT "Since the "; TC$ ;" is not new, maintenance cost
4350     PRINT "data will NOT be accurate. This is because
4360     PRINT "maintenance records were not kept up to ";
4370     PRINT "this date."
4380     PRINT
4390 '
4400 REM                VERIFY DATA ENTERED ABOVE
4410 '
4420     PRINT
4430      PRINT "IS ABOVE DATA CORRECT? (Y/N/A) "; CHR$(143)
4440       GOSUB 5090                            'GET KEYSTROKE
```

Program continued

```
4450      IF K$="y" GOSUB 4510 : GOTO 1330
4460      IF K$="a" GOTO 1330                    'abort
4470      IF K$<>"n" GOTO 4440
4480      CLS : PRINT@128, "RE-ENTER THE DATA:"
4490       GOTO 4220
4500 '
4510 REM                    OPEN & WRITE FILES
4520 '
4530   PRINT : PRINT "<< SAVING DATA >>"
4540 '
4550    OPEN"O",1,F1$                   'FUEL DATA FILE
4560    OPEN"O",2,F2$                   'MAINT. DATA FILE
4570 '
4580    PRINT#1, TC$","LEFT$(TIME$,8)","CM;TGU;N1;CPM;TFC;
4590 '
4600    PRINT#2, N2 ;
4610 '
4620   FOR X = 1 TO N2
4630    PRINT#2, CHR$(34); MI$(X) ;CHR$(34)"," ;
4640     PRINT#2, CHR$(34); DD$(X) ; CHR$(34) ; MC(X);
4650       NEXT X
4660 '
4670   FOR X = 1 TO N1
4680    PRINT#1, MPG(X) ;
4690     NEXT X
4700 '
4710    PRINT#2, MD ; TM
4720 '
4730    CLOSE
4740     RETURN
4750 '
4760 REM                OPEN & READ DATA FILES
4770 '
4780    CLS : PRINT@401, "<< READING DATA FROM DISK >>"
4790 '
4800    OPEN"I",1, F1$                  'FUEL DATA FILE
4810    OPEN"I",2, F2$                  'MAINTENANCE DATA FILE
4820 '
4830    INPUT#1, TC$, D$, CM, TGU, N1, CPM, TFC
4840    INPUT#2, N2
4850 '
4860    FOR X = 1 TO N2
4870     INPUT#2, MI$(X), DD$(X), MC(X)
4880       NEXT X
4890 '
4900   FOR X = 1 TO N1
4910    INPUT#1, MPG(X)
4920      NEXT X
4930 '
4940    INPUT#2, MD, TM
4950 '
4960    CLOSE : RETURN
4970 '
4980 REM                SCREEN-PAUSE SUBROUTINE
4990 '
5000    PRINT@960, "STRIKE <N> FOR NEXT SCREEN, <X> TO EXIT " ;
5010    PRINT CHR$(143) ;
5020 '
5030    GOSUB 5090                              'GET KEYSTROKE
5040 '
5050    IF K$="n" THEN CLS : RETURN
5060    IF K$<>"x" THEN 5030
5070      GOTO 1330
5080 '
5090 REM             INKEY$ SUBROUTINE
5100 '
5110   K$=INKEY$ : IF K$="" THEN 5110 ELSE RETURN
5120 '
5130 REM        SUBROUTINE TO CALCULATE NO. OF DOTS TO PRINT
5140 '
```

```
5150    X=47-LEN(P$)
5160    P$=P$+STRING$(X,".")
5170    RETURN
```

Celestial Software

by Michael J. Mangieri

One of the more perplexing problems in observing a celestial object is knowing where to look for it. Simple things, like the height and direction of the object, can be difficult to determine. Most of the good astronomy magazines and books on the market today do a fairly good job of listing celestial positions for the most common objects, and sometimes even have diagrams showing positions of the planets in altitude and azimuth. The diagrams, however, are usually for planets near the horizon at sunset or dawn and are almost always just for 40 degrees north latitude.

Armed with some good star atlases, astronomy texts, and the Celestial Software program, I can easily locate any object no matter where I may be or at what time of day I look.

The program runs on a 16K Level II TRS-80 and was designed to operate with an Okidata Microline-80 printer. The program:

- Converts celestial coordinates, right ascension, and declination (RA and DEC), to horizon coordinates, altitude, and azimuth (ALT and AZ), and vice versa.
- Works for any location in the world.
- Accurately computes the positions of any object for any date within the period from January 1, 1900 to January 1, 2100 to within 1/2 degree.
- Is menu driven.
- Produces plots and printed data.

Using the Program

Answer the memory size question with 32600. This leaves sufficient room for the assembly routine that generates the plots to the printer and for a keyboard debounce routine. I usually use KBEEPFIX by Dennis Kitsz and Block Cursor by Ron Balewski so the value of 32600 is appropriate.

After you load the program and type RUN, the title appears, followed by a brief pause while some data statements are collected and sorted. The program then asks for longitude (west), latitude, and a time zone correction.

Enter your longitude in degrees, minutes, and seconds, separated by periods, and with leading zeros included. Next, enter your latitude, using a plus sign for north and a minus sign for south. Finally, enter a time zone correction, which is a number used to convert local time into Universal time, which is used throughout the program. You can compute the value by

dividing your longitude by 15 and using only the integer portion of the result. For example, if your longitude is 77 degrees west, your time zone correction is 5 (77/15 = 5.1333). If you wish to enter time quantities in daylight saving time, subtract 1 from the time correction before entering the value. If you wish to use Universal time, enter 0.

After this, the program displays a set of instructions which give the format you must use to enter data into the program. Note that there are two ways to enter dates, and that time values are entered using a military clock format (00:00:00 to 24:00:00).

Press ENTER to see the menu. There are 12 options available in Celestial Software—10 for calculations of positions and two for plots and displays of the results. To select an option, press the up or down arrow to point to the option you want and then press S to select that option.

The 10 basic options are divided into two groups. Group 1, options 1 through 5, allows you to convert celestial coordinates to earth coordinates. Group 2, options 6 through 10, does the reverse.

Either HOLD, VARY, or a dotted line appears under each heading. The dotted line indicates the parameter being solved for. VARY indicates the parameter that changes with each entry. HOLD indicates the item that remains constant for all calculations.

After you select an option, the program displays that option at the top of the screen and asks for any constant data. Following this, the main display appears, and the program prompts you for the varying quantities. As you enter each item, the program displays the value inside the boxed area on the display that corresponds to the category of the entered item. When you have entered all required data for the first entry, the program computes the position values and displays them, then asks for the next entry. You can have up to 25 entries for each selection. To exit an option, type X.

If you make an illegal entry, such as a number not in the proper format for input, the program asks for the number again. If you wish to repeat a value from the previous entry, press ENTER without a number.

When you type X, the program returns you to the menu. You can then select a plot of the results, a display of results, or a new option. If you select a new option, all data for the previous option is lost.

Option 11 is the plot option. The program asks if you wish the resulting plot to be sent to a printer. Answer Y if you have a printer capable of printing TRS-80 graphics characters.

The next prompt is a choice between an auto plot and a manual plot. The auto plot selection allows the program logic to select the end points for each axis. The manual selection allows you to select your own ranges for altitude and azimuth. Valid ranges are:

Altitude	+90 to −90 degrees
Azimuth	+0 to 360 degrees

It is possible to enter a larger value for the starting point of the azimuth axis than for the ending point. This lets you shift the plot left or right so that points near due north appear contiguous. After the plot is completed, press any key to return to the menu.

AL$	String representation of altitude in degrees
AL( )	Numeric value of altitude in radians
AZ$	String representation of azimuth
AZ( )	Numeric value of azimuth in radians
D1$	String value of all entered data
DE$	String representation of declination in degrees
DE( )	Numeric value of declination in radians
DM( )	Day of the month
DR	Radians per degree
DY( )	Day of the year
ER	Error flag
HA	Hour angle
I	Index to all data tables
K	Constant used in calculation of sidereal time
LA	Latitude in radians
LA$	String representation of latitude
LO	Longitude in radians
LO$	String representation of longitude
MO( )	Month
RA( )	Right ascension
RA$	String representation of right ascension
RD	PI/2
ST( )	Sidereal time (hours)
TI$	Title for printer
TI( )	Time in hours
T1$	Time (string representation)
YR( )	Year
ZD	Zenith distance

Table 1. *Program variables*

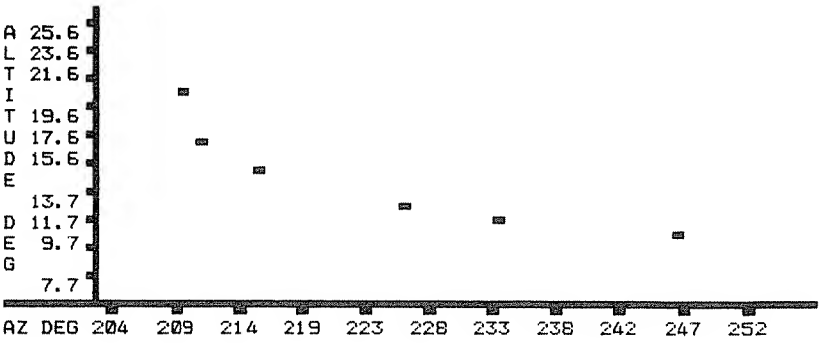
Option 12 provides a detailed report of any or all of the values calculated in the previous option. After you select option 12, the program asks you for a destination for the results. There are three possible choices—screen only, printer only, or both. This section of the program uses INKEY$ for entered data, so a set of instructions appears to tell you how to select data. If you select option 2 or 3, both of which activate the printer, you can have a title printed as well as the data. A title of about 15 to 25 characters centers neatly on the page.

The program traps most illegal entries. If, however, you make a mistake and have to press BREAK or reset the system, GOTO 240 should get you back to the menu portion of the program without a loss of data.

Table 1 lists the variables used by the program. Table 2 lists the major subroutines. Each option calls various subroutines to calculate data and display results. This makes the program easy to debug. Figure 1 shows an example of an auto plot followed by an option 12 request. Figure 2 is a sample of the same data with a manual plot selection of 0 to 90 degrees altitude and 0 to 360 degrees azimuth.

540	Enters data and time
2340	Converts RA$ to RA( )
2380	Converts DE$ to DE( )
2440	Formats RA$ and DE$ for display
2460	Displays altitude and azimuth (AL,AZ)
2480	Displays right ascension and declination (RA,DEC)
2510	Displays date
2520	Displays time
2530	Formats the screen for all displays
2620	Calculates altitude and azimuth (AL,AZ)
2720	Calculates right ascension and declination
2810	Calculates local sidereal time
2880	Accepts string representation of date and breaks it down into YR( ), MO( ), and DY( )
3050	Converts string representation of time into TI( )
3110	Converts hours into hours, minutes, and seconds
3200	SIN^{-1} and COS^{-1} calculation
3660	Plots axis
4130	Instructions display
4250,4260	Clear bottom of screen
4270	Displays line for option heading
4280	Printer routine—accepts title and prints site location (longitude and latitude). Also sets printer size.

Table 2. *Subroutines*



****** VENUS 30 MIN AFTER SUNSET ******

FOR THE LOCATION GIVEN BY THE FOLLOWING COORDINATES :

LONGITUDE = 077.00.00
LATITUDE = +39.00.00

SELECTION # 1
DATE --- 9 / 10 / 1981
TIME --- 18:52:60
RA ----- 13:37:48
DEC ---- -10:38:42
ALT ---- 10.7315 DEG
AZI ---- 246.714 DEG

SELECTION # 2
DATE --- 9 / 30 / 1981
TIME --- 18:22:00
RA ----- 15:07:12
DEC ---- -19:19:48
ALT ---- 11.4787 DEG
AZI ---- 233.203 DEG

SELECTION # 3
DATE --- 10 / 10 / 1981
TIME --- 18:02:60
RA ----- 15:53:56
DEC ---- -22:39:56
ALT ---- 12.8043 DEG
AZI ---- 226.174 DEG

SELECTION # 4
DATE --- 10 / 31 / 1981
TIME --- 17:36:36
RA ----- 17:34:00
DEC ---- -26:35:36
ALT ---- 15.0942 DEG
AZI ---- 215.417 DEG

SELECTION # 5
DATE --- 11 / 10 / 1981
TIME --- 17:22:60

```
RA ------    18:20:30
DEC ----    -26:49:48
ALT ----     17.0928 DEG
AZI ----     211.067 DEG


SELECTION #  6
DATE ---     11 / 30 / 1981
TIME ---    17:13:00
RA ------    19:42:30
DEC ----    -24:24:24
ALT ----     20.5656 DEG
AZI ----     209.348 DEG
```

Figure 1

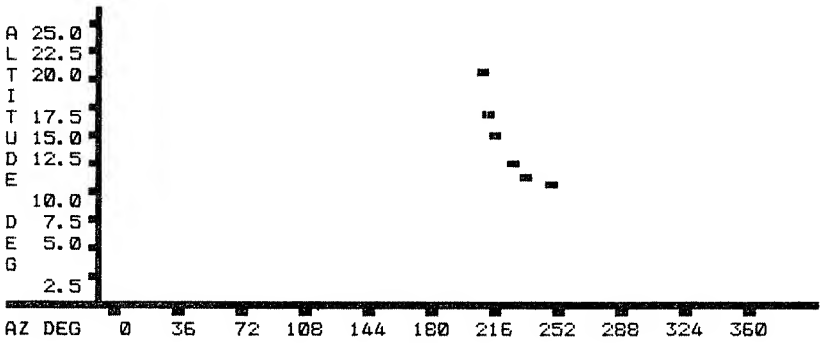


Figure 2

Program Listing. *Celestial Software*

```
10 REM **  CELESTIAL SOFTWARE I  VER 2.2  (05/29/81) ****
20 CLS:PRINT@402,"CELESTIAL  SOFTWARE  I";:PRINT@520,"CELESTIAL-TO-HOR
     IZON COORDINATE CONVERSION";:FORZ=1TO1000:NEXT:CLEAR175
30 DIM D9(12),AL(24),AZ(24),RA(24),DE(24),TI(24),ST(24),YR(24),DM(24),
     MO(24),DY(24)
40 M1$="###.#":M2$="###":PI=3.14159:RD=1.57079:DR=.017453:A#=23925.835
     9375:B#=8640184.542:C#=.0929
50 ONERRORGOTO4360
60 DD$="JANFEBMARAPRMAYJUNJULAUGSEPOCTNOVDEC"
70 FORZ=1TO12:READD9(Z):NEXT
80 FORZ=32604TO32634:READX:POKEZ,X:NEXT:POKE16526,92:POKE16527,127
90 CLS:PRINT@85,"PROGRAM INITIALIZATION"
100 GOSUB4270
110 PRINT:INPUT"LONGITUDE  (DDD.MM.SS)";LO$
120 IF LEN(LO$)<9 THEN 110
130 A$=LEFT$(LO$,3):A=VAL(A$):B$=MID$(LO$,5,2):B=VAL(B$)
140 C$=RIGHT$(LO$,2):C=VAL(C$)
150 LO=(A+B/60+C/3600)*DR
160 PRINT:INPUT"LATITUDE  (+/-DD.MM.SS)";LA$
170 IF LEN(LA$)<9 THEN 160
180 A$=LEFT$(LA$,3):A=VAL(A$):B$=MID$(LA$,5,2):B=VAL(B$)
190 C$=RIGHT$(LA$,2):C=VAL(C$)
200 C=B/60+C/3600:IF A<0 THEN LA=A-C ELSE LA=A+C
210 LA=LA*DR
220 PRINT:INPUT"ENTER TIME ZONE CORRECTION --- ";TZ
230 GOSUB 4130
240 CLS
250 PRINTTAB(7)"OPTION"TAB(19)"RA/DEC"TAB(31)"ALT/AZ"TAB(43)"DATE"TAB(
     55)"TIME":PRINT
260 PRINTTAB(10)"1"TAB(19)"VARY"TAB(31)"----"TAB(43)"HOLD"TAB(55)"HOLD
     "
270 PRINTTAB(10)"2"TAB(19)"VARY"TAB(31)"----"TAB(43)"VARY"TAB(55)"VARY
     "
280 PRINTTAB(10)"3"TAB(19)"HOLD"TAB(31)"----"TAB(43)"VARY"TAB(55)"HOLD
     "
290 PRINTTAB(10)"4"TAB(19)"HOLD"TAB(31)"----"TAB(43)"HOLD"TAB(55)"VARY
     "
300 PRINTTAB(10)"5"TAB(19)"HOLD"TAB(31)"----"TAB(43)"VARY"TAB(55)"VARY
     "
310 PRINTTAB(10)"6"TAB(19)"----"TAB(31)"VARY"TAB(43)"HOLD"TAB(55)"HOLD
     "
320 PRINTTAB(10)"7"TAB(19)"----"TAB(31)"VARY"TAB(43)"VARY"TAB(55)"VARY
     "
330 PRINTTAB(10)"8"TAB(19)"----"TAB(31)"HOLD"TAB(43)"VARY"TAB(55)"HOLD
     "
340 PRINTTAB(10)"9"TAB(19)"----"TAB(31)"HOLD"TAB(43)"HOLD"TAB(55)"VARY
     "
350 PRINTTAB(9)"10"TAB(19)"----"TAB(31)"HOLD"TAB(43)"VARY"TAB(55)"VARY
     "
360 PRINTTAB(9)"11"TAB(19)"************      PLOT      ************"
370 PRINTTAB(9)"12"TAB(19)"************     DISPLAY     ************"
380 PRINT:PRINTTAB(09)"KEY:       [ = UP       "CHR$(92)" = DOWN
      S = SELECT";
390 A$="    --->":C$="         "
400 PRINT@A9+128,A$;
410 B$=INKEY$:IF B$="[" THEN 450
420 IF PEEK(14400)=16 THEN 480
430 IF B$="S" THEN 510
440 GOTO 410
450 PRINT@A9+128,C$;:A9=A9-64:IF A9<0 THEN A9=0
460 PRINT@A9+128,A$;:IF PEEK(14400)=8 THEN FORZ=1TO50:NEXT:GOTO450
470 GOTO 410
480 PRINT@A9+128,C$;:A9=A9+64:IF A9>704 THEN A9=704
490 PRINT@A9+128,A$;:IF PEEK(14400)=16 THEN FORZ=1TO50:NEXT:GOTO 480
500 GOTO 410
510 ON(A9/64+1)GOTO 520,750,950,1150,1340,1510,1660,1840,2010,2180,323
     0,3730
```

```
520 CLS:I=0:PRINTTAB(25)"OPTION  1"
530 GOSUB4270:GOSUB540:GOTO590
540 PRINT:INPUT"DATE  ( MON DY YEAR   OR  MM.DD.YY )";D1$
550 GOSUB2880:IF ER=1 THEN 540
560 PRINT:INPUT"TIME  ( HH.MM.SS  IN  24 HR. FORMAT )";T1$
570 GOSUB 3050: IF ER=2 THEN 560
580 RETURN
590 CLS:PRINTCHR$(23):GOSUB2530:GOSUB2810:GOSUB2510:GOSUB2520
600 PRINT@840,"RA("I+1") = ";:INPUTRA$
610 IF RA$="X"THEN240
620 IF LEN(RA$)=8THEN640
630 GOSUB 4250:GOTO 600
640 GOSUB 2340:IF ER=3 THEN 630
650 PRINT@392,RA$;
660 PRINT@902,"DEC("I+1") = ";:INPUTDE$
670 IF LEN(DE$)=9 THEN 690
680 GOSUB 4260: GOTO 660
690 GOSUB 2380:IF ER=4 THEN 680
700 PRINT@422,DE$;
710 GOSUB 4250:GOSUB 4260
720 ST(I)=ST(0):TI(I)=TI(0):MO(I)=MO(0):DM(I)=DM(0):YR(I)=YR(0):GOSUB
     2620
730 GOSUB 2460
740 I=I+1:GOTO 600
750 CLS:I=0:PRINTTAB(25)"OPTION 2"
760 GOSUB4270
770 PRINT:PRINTTAB(10)"RA/DEC, DATE, AND TIME VARYING QUANTITIES"
780 FOR Z=0 TO 1000: NEXT:CLS
790 PRINTCHR$(23):GOSUB 2530
800 PRINT@840,"DATE ("I+1") = ";:INPUTD1$
810 IF D1$="X" THEN 240
820 GOSUB 2880:IF ER=1 THEN GOSUB4250:GOTO800
830 GOSUB 2510:PRINT@904,"TIME ("I+1") = ";:INPUTT1$
840 GOSUB 3050:IF ER=2 THEN GOSUB4260:GOTO830
850 GOSUB 2520:GOSUB 2810:GOSUB 4250:GOSUB 4260
860 PRINT@840,"RA ("I+1") = ";:INPUTRA$
870 IF LEN(RA$)<>8 THENGOSUB4250:GOTO860
880 GOSUB 2340:IF ER=3 THENGOSUB4250:GOTO860
890 PRINT@392,RA$;
900 PRINT@902,"DEC ("I+1") = ";:INPUTDE$
910 IF LEN(DE$)<>9THENGOSUB4260:GOTO900
920 GOSUB 2380:IF ER=4 THENGOSUB4260:GOTO900
930 PRINT@422,DE$;
940 GOSUB 4250:GOSUB 4260:GOSUB 2620:GOSUB 2460:I=I+1:GOTO800
950 CLS:I=0:PRINTTAB(25)"OPTION 3"
960 GOSUB4270
970 PRINTTAB(15)"RA ( HH.MM.SS )  = ";:INPUT RA$
980 IF LEN(RA$)<>8 THEN 970
990 GOSUB 2340:IF ER=3 THEN 970
1000 PRINTTAB(15)"DEC ( +/- DD.MM.SS ) = ";:INPUT DE$
1010 IF LEN(DE$)<>9 THEN 1000
1020 GOSUB 2380:IF ER=4 THEN 1000
1030 PRINT:PRINTTAB(15)"TIME ( HH.MM.SS  24HR FORMAT ) = ";:INPUT T1$
1040 GOSUB 3050:IF ER=2 THEN 1030
1050 CLS:PRINTCHR$(23):GOSUB 2530
1060 PRINT@102,B1$":"A1$":"C1$;
1070 PRINT@392,RA$;:PRINT@422,DE$;
1080 PRINT@840,"DATE ("I+1") = ";:INPUT D1$
1090 IF D1$="X" THEN 240
1100 GOSUB 2880:IF ER=1 THEN 1140
1110 TI(I)=TI(0):RA(I)=RA(0):DE(I)=DE(0)
1120 GOSUB 2510:GOSUB 2810
1130 GOSUB 2620:GOSUB 2460:I=I+1
1140 GOSUB 4250:GOTO1080
1150 CLS:I=0:PRINTTAB(25)"OPTION 4"
1160 GOSUB4270
1170 PRINTTAB(15)"RA ( HH.MM.SS ) = ";:INPUT RA$
1180 IF LEN(RA$)<>8 THEN 1170
1190 GOSUB 2340:IF ER=3 THEN 1170
1200 PRINTTAB(15)"DEC (+/-DD.MM.SS ) = ";:INPUT DE$
```

Program continued

```
1210 IF LEN(DE$)<>9 THEN 1200
1220 GOSUB 2380: IF ER=4 THEN 1200
1230 PRINT:PRINTTAB(5)"DATE ( MMM DD YYYY  OR  MM.DD.YY ) = ";:INPUT D
     1$
1240 GOSUB 2880:IF ER=1 THEN 1230
1250 CLS:PRINTCHR$(23):GOSUB 2530
1260 GOSUB2510:PRINT@392,RA$;:PRINT@422,DE$;
1270 PRINT@840,"TIME ("I+1") = ";:INPUT T1$
1280 IF T1$="X" THEN 240
1290 GOSUB3050:IF ER=2 THEN 1330
1300 RA(I)=RA(0):DE(I)=DE(0):MO(I)=MO(0):DM(I)=DM(0):YR(I)=YR(0):DY(I)
     =DY(0)
1310 GOSUB 2520:GOSUB2810
1320 GOSUB2620:GOSUB2460:I=I+1
1330 GOSUB 4250:GOTO1270
1340 CLS:I=0:PRINTTAB(25)"OPTION 5"
1350 GOSUB4270
1360 PRINT:PRINTTAB(15)"RA ( HH.MM.SS ) = ";:INPUTRA$
1370 IF LEN(RA$)<>8 THEN 1360
1380 GOSUB2340:IF ER=3 THEN 1360
1390 PRINTTAB(15)"DEC (+/-DD.MM.SS ) = ";:INPUTDE$
1400 IF LEN(DE$)<>9 THEN 1390
1410 GOSUB2380:IF ER=2 THEN 1390
1420 CLS:PRINTCHR$(23):GOSUB2530:GOSUB2480
1430 PRINT@840,"DATE ("I+1") = ";:INPUTD1$
1440 IF D1$="X" THEN 240
1450 GOSUB2880:IF ER=1 THENGOSUB4250:GOTO1430
1460 GOSUB2510:PRINT@904,"TIME ("I+1") = ";:INPUTT1$
1470 GOSUB3050:IF ER=2 THENGOSUB4260:GOTO1460
1480 RA(I)=RA(0):DE(I)=DE(0)
1490 GOSUB2520:GOSUB2810:GOSUB4250:GOSUB4260
1500 GOSUB2620:GOSUB2460:I=I+1:GOTO1430
1510 CLS:I=0:PRINTTAB(25)"OPTION 6"
1520 GOSUB4270
1530 GOSUB540
1540 CLS:PRINTCHR$(23):GOSUB2530:GOSUB2810
1550 GOSUB2510:GOSUB2520
1560 PRINT@840,"ALT ("I+1") = ";:INPUT AL$
1570 IF AL$="X" THEN 240
1580 AL(I)=VAL(AL$)*DR
1590 PRINT@708,AL(I)/DR;
1600 PRINT@904,"AZI ("I+1") = ";:INPUT AZ$
1610 AZ(I)=VAL(AZ$)*DR
1620 PRINT@738,AZ(I)/DR;
1630 GOSUB4250:GOSUB4260:ST(I)=ST(0):TI(I)=TI(0):MO(I)=MO(0):DM(I)=DM(
     0):YR(I)=YR(0):GOSUB2720
1640 GOSUB2440:GOSUB2480
1650 I=I+1:GOTO1560
1660 CLS:I=0:PRINTTAB(25)"OPTION 7"
1670 GOSUB4270
1680 PRINT:PRINTTAB(10)"ALT/AZI, DATE, AND TIME VARYING QUANTITIES"
1690 FORZ=0TO1000:NEXT:CLS
1700 PRINTCHR$(23):GOSUB2530
1710 PRINT@840,"DATE ("I+1") = ";:INPUTD1$
1720 IFD1$="X"THEN240
1730 GOSUB2880:IFER=1THENGOSUB4250:GOTO1710
1740 GOSUB2510:PRINT@904,"TIME ("I+1") = ";:INPUTT1$
1750 GOSUB3050:IFER=2THENGOSUB4260:GOTO1740
1760 GOSUB2520:GOSUB2810:GOSUB4250:GOSUB4260
1770 PRINT@840,"ALT ("I+1") = ";:INPUTAL$
1780 AL(I)=VAL(AL$)*DR
1790 PRINT@708,AL(I)/DR;
1800 PRINT@904,"AZI ("I+1") = ";:INPUTAZ$
1810 AZ(I)=VAL(AZ$)*DR
1820 PRINT@738,AZ(I)/DR;
1830 GOSUB4250:GOSUB4260:GOSUB2720:GOSUB2440:GOSUB2480:I=I+1:GOTO1710
1840 CLS:I=0:PRINTTAB(25)"OPTION 8"
1850 GOSUB4270
1860 PRINTTAB(15)"ALT (DEGREES) = ";:INPUTAL$
1870 AL(I)=VAL(AL$)*DR
```

```
1880 PRINTTAB(15)"AZI (DEGREES) = ";:INPUTAZ$
1890 AZ(I)=VAL(AZ$)*DR
1900 PRINT:PRINTTAB(15)"TIME ( HH.MM.SS  24HR FORMAT ) = ";:INPUT T1$
1910 GOSUB 3050:IF ER=2 THEN 1900
1920 CLS:PRINTCHR$(23):GOSUB 2530
1930 PRINT@102,B1$":"A1$":"C1$;:PRINT@708,AL(I)/DR;:PRINT@738,AZ(I)/DR
     ;
1940 PRINT@840,"DATE ("I+1") = ";:INPUT D1$
1950 IF D1$="X" THEN 240
1960 GOSUB 2880:IF ER=1 THEN 2000
1970 TI(I)=TI(0):AZ(I)=AZ(0):AL(I)=AL(0)
1980 GOSUB 2510:GOSUB 2810
1990 GOSUB2720:GOSUB2440:GOSUB 2480:I=I+1
2000 GOSUB 4250:GOTO1940
2010 CLS:I=0:PRINTTAB(25)"OPTION 9"
2020 GOSUB4270
2030 PRINTTAB(15)"ALT (DEGREES) = ";:INPUT AL$
2040 AL(I)=VAL(AL$)*DR
2050 PRINTTAB(15)"AZI (DEGREES) = ";:INPUT AZ$
2060 AZ(I)=VAL(AZ$)*DR
2070 PRINT:PRINTTAB(5)"DATE ( MMM DD YYYY  OR  MM.DD.YY ) = ";:INPUTD1
     $
2080 GOSUB 2880:IF ER=1 THEN 2070
2090 CLS:PRINTCHR$(23):GOSUB 2530
2100 GOSUB 2510:PRINT@708,AL(I)/DR;:PRINT@738,AZ(I)/DR;
2110 PRINT@840,"TIME ("I+1") = ";:INPUT T1$
2120 IF T1$="X" THEN 240
2130 GOSUB 3050:IF ER=2 THEN 2170
2140 AL(I)=AL(0):AZ(I)=AZ(0):DY(I)=DY(0):DM(I)=DM(0):YR(I)=YR(0):MO(I)
     =MO(0)
2150 GOSUB 2520:GOSUB 2810
2160 GOSUB 2720:GOSUB 2440:GOSUB 2480:I=I+1
2170 GOSUB 4250:GOTO 2110
2180 CLS:I=0:PRINTTAB(25)"OPTION 10"
2190 GOSUB4270
2200 PRINTTAB(15)"ALT (DEGREES) = ";:INPUTAL$
2210 AL(I)=VAL(AL$)*DR
2220 PRINTTAB(15)"AZI (DEGREES) = ";:INPUTAZ$
2230 AZ(I)=VAL(AZ$)*DR
2240 CLS:PRINTCHR$(23):GOSUB2530:GOSUB2460
2250 PRINT@840,"DATE ("I+1") = ";:INPUTD1$
2260 IFD1$="X"THEN240
2270 GOSUB2880:IFER=1THENGOSUB4250:GOTO2250
2280 GOSUB2510
2290 PRINT@904,"TIME ("I+1") = ";:INPUTT1$
2300 GOSUB3050:IFER=2THENGOSUB4260:GOTO2290
2310 AL(I)=AL(0):AZ(I)=AZ(0)
2320 GOSUB2520:GOSUB4250:GOSUB4260
2330 GOSUB2810:GOSUB2720:GOSUB2440:GOSUB2480:I=I+1:GOTO2250
2340 A=VAL(LEFT$(RA$,2)):B=VAL(MID$(RA$,4,2)):C=VAL(RIGHT$(RA$,2))
2350 RA(I)=A+B/60+C/3600:ER=0
2360 IF RA(I)<0 OR RA(I)>24 THENER=3
2370 RETURN
2380 S$=LEFT$(DE$,1):ER=0
2390 A=VAL(MID$(DE$,2,2)):B=VAL(MID$(DE$,5,2)):C=VAL(RIGHT$(DE$,2))
2400 D=(A+B/60+C/3600)*DR:DE(I)=D
2410 IF S$="-" THEN DE(I)=-DE(I)
2420 IF DE(I)<-RD OR DE(I)>RD THENER=4
2430 RETURN
2440 A=RA(I):GOSUB3110:RA$=A$
2450 A=DE(I)/DR:GOSUB3110:DE$=A$:RETURN
2460 PRINT@708,STRING$(27," ");
2470 PRINT@708,AL(I)/DR;:PRINT@738,AZ(I)/DR;:RETURN
2480 PRINT@392,STRING$(26," ");
2490 PRINT@392,RA$;:PRINT@422,DE$;
2500 PRINT@708,STRING$(28," ");:RETURN
2510 PRINT@68,"              ";:PRINT@70,D1$;:RETURN
2520 PRINT@100,"          ";:PRINT@102,B1$":"A1$":"C1$;:RETURN
2530 PRINT@14,"DATE";:PRINT@42,"TIME";
2540 PRINT@128,STRING$(32,"-")
```

Program continued

```
2550 PRINT@270,"RA";:PRINT@300,"DEC";
2560 PRINT@324,"------------";:PRINT@354,"------------";
2570 PRINT@452,"------------";:PRINT@482,"------------";
2580 PRINT@590,"ALT";:PRINT@620,"AZI";
2590 PRINT@644,"------------   ------------";
2600 PRINT@772,"------------   ------------";
2610 RETURN
2620 IF RA(I)=24 THEN RA(I)=0
2630 HA=(ST(I)-RA(I))*DR*15
2640 ZD=SIN(LA)*SIN(DE(I))+COS(LA)*COS(DE(I))*COS(HA)
2650 A=1:B=ZD:GOSUB 3200:ZD=B:AL(I)=PI/2-ZD
2660 A3=TAN(LA)*TAN(AL(I))
2670 A4=SIN(DE(I))/COS(LA)/COS(AL(I))
2680 A5=A4-A3:A=1:B=A5:GOSUB3200
2690 IFHA<0THENAZ(I)=BELSEAZ(I)=2*PI-B
2700 IFHA<-PIORHA>PITHENAZ(I)=2*PI-AZ(I)
2710 RETURN
2720 ZD=PI/2-AL(I)
2730 D=SIN(LA)*COS(ZD)+COS(LA)*SIN(ZD)*COS(AZ(I))
2740 A=2:B=D:GOSUB 3200:DE(I)=B
2750 B3=TAN(LA)*TAN(DE(I))
2760 B4=COS(ZD)/COS(LA)/COS(DE(I))
2770 B5=B4-B3:A=1:B=B5:GOSUB3200:HA=B
2780 HA=HA/DR/15:IF AZ(I)/DR-180<0 THEN HA=-HA
2790 RA(I)=ST(I)-HA:IF RA(I)<0 THEN RA(I)=24+RA(I)
2795 IFRA(I)>24THENRA(I)=RA(I)-24
2800 RETURN
2810 D#=YR(I)-1900:E#=INT(D#/4)+D#*365-.5:T#=E#/36525.0
2820 S#=A#+B#*T#+C#*T#*T#:S#=S#/86400.0:S#=(S#-INT(S#))*24
2830 IF INT(D#/4)-D#/4=0,S#=S#-.0657098
2840 K=S#:N=DY(I):T=TI(I)+TZ
2850 ST(I)=K+0.0657*N+1.0027*T-(LO/15/DR)
2860 IF ST(I)>24 THEN ST(I)=ST(I)-24
2870 RETURN
2880 D$=D1$:ER=0
2890 IF VAL(LEFT$(D$,2))=0 THEN 2940
2900 A$=MID$(D$,1,2):MO(I)=VAL(A$):A$=MID$(D$,4,2):DM(I)=VAL(A$)
2910 A$=MID$(D$,7,2):A$="19"+A$:IF VAL(A$)=19THEN2920ELSEYR(I)=VAL(A$)
     :GOTO3000
2920 YR(I)=YR(I-1)
2930 GOTO 3000
2940 A$=MID$(D$,1,3)
2950   FORZ=1TO34STEP3:IFA$=MID$(DD$,Z,3)THEN2970
2960   NEXT:ER=1:RETURN
2970 MO(I)=Z/3+2/3
2980 A$=MID$(D$,5,2):DM(I)=VAL(A$)
2990 A$=MID$(D$,8,4):YR(I)=VAL(A$):IFYR(I)=0THENYR(I)=YR(I-1)
3000 A=D9(MO(I)):A=A+DM(I)
3010 IF MO(I)>2  AND ((YR(I)/4)-FIX(YR(I)/4))=0 THEN A=A+1
3020 DY(I)=A
3030 IF MO(I)<0 OR MO(I)>12 OR YR(I)<1900 THEN ER=1
3040 RETURN
3050 A$=MID$(T1$,4,2):A1$=A$
3060 B$=LEFT$(T1$,2):B1$=B$
3070 C$=RIGHT$(T1$,2):C1$=C$:ER=0
3080 TI(I)=VAL(B$)+VAL(A$)/60+VAL(C$)/3600
3090 IF TI(I)<0 OR TI(I)>24 THENER=2
3100 RETURN
3110 IF A>=0 THEN 3130
3120 A=-A:SG=1
3130 B=INT(A):C=(A-B)*60:D=INT(C):E=INT((C-D)*60+.5)
3140 A$=RIGHT$(STR$(B),2):B$=RIGHT$(STR$(D),2):C$=RIGHT$(STR$(E),2)
3150 IFLEFT$(B$,1)=" ",B$="0"+RIGHT$(B$,1)
3160 IFLEFT$(C$,1)=" ",C$="0"+RIGHT$(C$,1)
3170 A$=A$+":"+B$+":"+C$
3180 IF SG=1 THEN A$="-"+A$
3190 SG=0:RETURN
3200 IF A=1 THEN 3220
3210 B=ATN(B/SQR(-B*B+1)):RETURN
3220 B=RD-ATN(B/SQR(-B*B+1)):RETURN
```

```
3230 CLS:PRINTTAB(22)"OPTION 11 -- PLOT"
3240 GOSUB4270
3250 PRINTTAB(15)"PLOT TO PRINTER  (Y/N)";:INPUTQ$
3260 IFQ$="Y"THENPG=1:LPRINTCHR$(27)CHR$(66):ELSEPG=0
3270 PRINTTAB(15)"AUTO PLOT  ( Y/N )";:INPUT Q$
3280 IF Q$<>"N" THEN 3570
3290 PRINT:PRINT@330,"MIN ALT = ";:INPUT Y1
3300 PRINT@350,"MAX ALT = ";:INPUT Y2
3310 PRINT@458,"MIN AZI = ";:INPUT X1
3320 PRINT@478,"MAX AZI = ";:INPUT X2
3330 IFX2<X1THENX2=X2+360
3340 GOSUB 3660
3350 DY=(Y2-Y1):DX=(X2-X1):D3=DY/10:D4=DX/10
3360 FOR Z=1 TO 10
3370 IF Z=1 THEN PRINT@833,USINGM1$;D3*Z+Y1;
3380 IF Z>=2 AND Z<5 THEN PRINT@(833-64*Z),USINGM1$;D3*Z+Y1;
3390 IF Z>=5 AND Z<8 THEN PRINT@(769-64*Z),USINGM1$;D3*Z+Y1;
3400 IF Z>=8 AND Z<11THEN PRINT@(705-64*Z),USINGM1$;D3*Z+Y1;
3410 NEXT
3420 FOR Z=0 TO 10
3430 D8=D4*Z+X1:IF D8<0 THEN D8=360-D8
3440 IF D8>360 THEN D8=D8-360
3450 PRINT@(5*Z+967),USINGM2$;D8;
3460 NEXT
3470 FOR Z=0 TO I-1
3480 X=(100/(X2-X1))*(AZ(Z)/DR)-(100*X1)/(X2-X1)+16
3490 IFAZ(Z)/DR<X1ANDX2>360THENX=X+36000/(X2-X1)
3500 Y=(-40/(Y2-Y1))*(AL(Z)/DR)+(40*Y1)/(Y2-Y1)+42
3510 X=INT(X+.5):Y=INT(Y+.5)
3520 IF X<16 OR X>116 THEN 3550
3530 IF Y>42 OR Y<2 THEN 3550
3540 SET(X,Y):SET(X+1,Y)
3550 NEXT:IFPG,PRINTUSR(0)
3560 Z$=INKEY$:IF Z$=""THEN 3560 ELSE 240
3570 Z1=0:Z2=2*PI:Z3=0:Z4=PI/4
3580 FOR Z=0 TO I-1
3590 IF AZ(Z)>Z1 THEN Z1=AZ(Z)
3600 IF AZ(Z)<Z2 THEN Z2=AZ(Z)
3610 IF AL(Z)>Z3 THEN Z3=AL(Z)
3620 IF AL(Z)<Z4 THEN Z4=AL(Z)
3630 NEXT
3640 X1=Z2/DR-5:X2=Z1/DR+5:Y1=Z4/DR-5:Y2=Z3/DR+5
3650 GOTO 3340
3660 CLS:FOR Z=0 TO 42:SET(14,Z):NEXT
3670 FOR Z=0 TO 127:SET(Z,42):NEXT
3680 FORZ=1TO11:SET(13,4*Z-2):SET(10*Z+6,43):SET(10*Z+7,43):NEXT
3690 Q$="ALTITUDE DEG"
3700 FORZ=1TO12:PRINT@(64*Z),MID$(Q$,Z,1);:NEXT
3710 PRINT@960,"AZ DEG";
3720 RETURN
3730 CLS:Z1$=""
3740 PRINTTAB(15)"SELECT OPTION. . ."
3750 PRINT:PRINTTAB(20)"1. PRINT TO SCREEN"
3760 PRINTTAB(20)"2. PRINT TO LINE PRINTER"
3770 PRINTTAB(20)"3. PRINT TO BOTH"
3780 Q$=INKEY$:IFQ$=""THEN3780
3790 PS=VAL(Q$)
3800 IFPS=1THEN3810ELSEGOSUB4280
3810 PRINT:PRINT"FOR DATA ON EACH ENTRY, ENTER SELECTION NUMBER FOLLOW
     ED"
3820 PRINT"BY A '.'; ENTER 'X.' TO END."
3830 PRINT"ENTER '0.' TO PRINT ALL ENTRIES"
3840 Z1$=INKEY$:IFZ1$=""THEN3840
3850 GOTO 4100
3860 IF Z1$="X" THEN 240 ELSE Z=VAL(Z1$)-1
3870 Z1$="":IFZ=-1THENL1=0:L2=I-1:ELSEL1=Z:L2=Z
3880 FORZ=L1TOL2
3890 A=TI(Z):GOSUB 3110:AB$=A$
3900 A=RA(Z):GOSUB 3110:AC$=A$
3910 A=DE(Z)/DR:GOSUB 3110:AD$=A$
```

Program continued

```
3920 CLS:PRINTCHR$(23)
3930 IFPS<>2THENPRINT:PRINT"SELECTION # ";Z+1
3940 IFPS<>1THENLPRINT"SELECTION # ";Z+1
3950 IFPS<>2THENPRINT:PRINT"DATE ---    ";MO(Z)"/"DM(Z)"/"YR(Z)
3960 IFPS<>1THENLPRINT:LPRINT"DATE ---    ";MO(Z)"/"DM(Z)"/"YR(Z)
3970 IFPS<>2THENPRINT"TIME ---    ";AB$
3980 IFPS<>1THENLPRINT"TIME ---    ";AB$
3990 IFPS<>2THENPRINT"RA -----    ";AC$
4000 IFPS<>1THENLPRINT"RA -----    ";AC$
4010 IFPS<>2THENPRINT"DEC ----    ";AD$
4020 IFPS<>1THENLPRINT"DEC ----    ";AD$
4030 IFPS<>2THENPRINT"ALT ----    ";AL(Z)/DR;"DEG"
4040 IFPS<>1THENLPRINT"ALT ----    ";AL(Z)/DR;"DEG"
4050 IFPS<>2THENPRINT"AZI ----    ";AZ(Z)/DR;"DEG"
4060 IFPS<>1THENLPRINT"AZI ----    ";AZ(Z)/DR;"DEG"
4070 IFPS<>1THENLPRINT" ":LPRINT" "
4080 NEXT
4090 PRINT:PRINT"ENTER NEXT SELECTION # OR X";
4100 Z$=INKEY$:IF Z$="" THEN 4100
4110 IF Z$="." THEN 3860
4120 Z1$=Z1$+Z$:GOTO4100
4130 CLS:PRINTTAB(26)"INSTRUCTIONS"
4140 GOSUB4270
4150 PRINT"    THE SYSTEM WILL AUTOMATICALLY PROMPT FOR ALL NECESSARY"

4160 PRINT"ENTRIES REQUIRED FOR THE SELECTED OPTION.  ENTRIES MUST BE
     IN "
4170 PRINT"THE FOLLOWING FORMATS:"
4180 PRINT:PRINT"     RA . . . .  HH.MM.SS        DEC . . . .  +/-DD.M
     M.SS"
4190 PRINT"     ALT. . . .  DEGREES         AZI . . . .  DEGREES"
4200 PRINT"     DATE . . .  MMM DD YYYY      OR      MM.DD.YY"
4210 PRINT"     TIME . . .  HH.MM.SS  IN  24HR CLOCK FORMAT"
4220 PRINT:PRINT"     WHEN YOU WISH TO EXIT THE SELECTED OPTION, TYPE
     'X'"
4230 PRINT:PRINTTAB(21)"PRESS ENTER TO START";
4240 Q$=INKEY$:IF Q$=""THEN4240ELSERETURN
4250 PRINT@838,STRING$(48," ");:RETURN
4260 PRINT@902,STRING$(48," ");:RETURN
4270 PRINTSTRING$(64,"-"):RETURN
4280 INPUT"TITLE FOR PRINTER HEADING?  ";TI$:LPRINTCHR$(27)CHR$(66)
4290 LPRINTTAB(15)"*****  ";TI$;"  *****"
4300 LPRINTSTRING$(2,128)
4310 LPRINT"FOR THE LOCATION GIVEN BY THE FOLLOWING COORDINATES :"
4320 LPRINTSTRING$(1,128):LPRINT"LONGITUDE = ";LO$:LPRINT"LATITUDE  =
     ";LA$:LPRINTSTRING$(2,128):RETURN
4330 DATA 0,31,59,90,120,151,181,212,243,273,304,334
4340 DATA 245,197,213,229,33,0,60,22,16,6,64,78,205,141,5,35,16,249,14
     ,13,205,141,5,21,32,239,225,209,193,241,201
4350 PRINTSTRING$(64,"-"):RETURN
4360 REM *** ERROR TRAP ***
4370 IFERR/2+1=9,PRINT@840,"MAX ENTRIES EXCEEDED ";:I=25:GOTO4380:ELSE
     B=0:RESUMENEXT
4380 FORZ=1TO1000:NEXT:GOTO240
```

Your Personal Expense Account

by D. J. Kelly

For the last 20 years, I have been engaged in two professional activities that have substantial deductible income tax expenses which must be sorted out and reported separately. During this time, I have found it important to keep track of *all* expenses without regard to their potential deductibility. Many times, what I had thought was not deductible at the first of the year turned out to be a valuable deduction (or would have been had I kept track of the expenditure). As a result, I developed this expense account program that is effective and simple to use. This program doesn't need an accompanying textbook and doesn't produce or need data tapes.

The keystone of any worthwhile expense account program is keeping receipts for every expenditure (or at least as many as possible). You must also annotate the receipts as to what was purchased in order to categorize them properly. If receipts were not given, or were lost, make a handwritten note of the transaction particulars, such as the date, what was purchased, the cost, and where it was purchased. For transactions in which receipts are not normally given, such as the cash purchase of gasoline, no receipt is necessary to substantiate a tax deduction. The general rule is this: Keep every receipt (or make one), regardless of its potential tax deductibility!

The Program

The prerequisites for this program are:

- 16K memory, minimum
- Level II BASIC
- Video display
- Printer (see note at end of text)

As written, the program provides 40 categories numbered 0 to 39. (See Program Listing.) You can change this number to suit your needs by changing line 15 (LET BA = 39). The variable BA is used to define the two DIM statements (lines 20 and 30) as well as the upper limit of the six FOR-NEXT loops which enable the program to store the values for each category (lines 31, 1010, 2010, 2510, 3040, and 11070). Table 1 shows the categories I use. The program maintains two registers of expense categories:

Register A	Monthly totals. This register returns to zero at the start of each new month run.
Register B	Cumulative totals. This register records the cumulative amounts for each category as long as the program is being worked.

Work Away from Home Codes

1 Lodging	2 Utilities
3 Food and meals	5 Telephone—see writing
29 Gas and oil	30 Service and repairs

Writing Codes

5 Telephone (split with professional)	16 Postage
17 Stationery and supplies	18 Equipment
19 Subscriptions, books, publications	20 Fees (agent, etc.)
21 Travel	22 Copies, research, tests
23 Post office box rent	39 Special projects

Professional and Work Codes

8 Misc. work search	9 Printed materials
10 Transportation	11 Meals (on the road)
12 Lodging (on the road)	13 Licenses
14 Prof. magazines, books, publications	15 Prof. equipment and supplies

Medical Codes

24 Hospital	25 Insurance
26 Medicines	27 Doctor calls
28 Misc. medical	

Alimony Payment Codes

35 Alimony

Living Expense Codes

6 Clothing	7 Furnishings
31 Car payments	Car license, insurance

Misc Codes

34 Entertainment (movies, records, etc.)	37 Depreciable equipment
0 Unaccounted expenses	

Table 1. *Category assignments*

Master Menu

Execution of a program run or completion of most of the subroutines causes the program to display the master menu (see Table 2). This menu gives a choice of eight subroutines and an option to end the program. Most of the subroutines are complete programs and are not dependent on another subroutine. Thus, they may be initiated at any time and in any sequence. Of course, logic dictates that you enter some data into the registers before asking the computer to list the contents, but the Summary to Date and the Year End Summary will run with all zeros.

SELECT ROUTINE DESIRED
1 = CARRY-OVER INPUTS
2 = NEW MONTH INPUTS
3 = SUMMATION TO DATE
4 = YEAR END SUMMATION
5 = FLAG ACCOUNTS
6 = REGISTER VALUE CORRECTION
7 = NOTES AND INSTRUCTIONS
8 = END PROGRAM
9 = SPECIAL DATA RECAPTURE

Table 2. *Master menu*

Carry-over Inputs (routine 1)

Since the program does not use data tapes, I included a way to enter data from previous program runs. This routine may be run as many times as necessary to bring register B up to where the previous run stopped. The routine prints the expense category number on the video display, starting with zero. Each time you enter a figure, it advances to the next number in sequence. When the figure for category 39 has been entered, the program returns to the master menu where you can initiate the routine again to enter more data.

There is no provision in this routine for a printout. If you type 3 when the control returns to the master menu, the Summation to Date routine is activated, and register B is printed out.

New Month Input (routine 2)

In this routine, each expenditure for the month is entered, and a printed record is produced. Data is entered in response to instructions printed on the video display. Paid To and Check Number are shown in one set and Date, Amount, and Expense Code are shown in the other set. The computer then adds the given amount to both registers for the category listed. This is followed by a printout of the complete transaction and assignment of a transaction number.

One unique feature of this program is the use of a string variable for the check number. This permits the use of letters to prefix a check number, in order to indicate different banks, money orders, cashier's checks, and so on. Another feature is the provision in lines 2110 through 2360 to assign letter codes to payees that appear often in your expense accounts. For instance, in my program, the letter A prints out as American Express.

I built an error correcting routine into the New Month routine. If you hit the ENTER key and realize you typed the wrong amount or expense code, just type ERROR and a comma when asked for the next Pay To and Check

Number. (Be sure to type the comma, or you'll get an incomplete response from the computer.) This entry causes the computer to print out ERROR IN ITEM X DELETE ENTRY then shift to the register correction routine where you can subtract the incorrect amount from the A and B registers. The Register Correction routine then gives you three options to its prompt OTHER REGISTER CORRECTIONS: Yes, to Return to Master Menu, or return to current month. You, of course, want Return to Current Month, where you can enter the correct data.

When all the month's receipts have been entered, type END. This causes the computer to print out a list of categories and the amounts for that month from register A, then return to the master menu. Figure 1 shows a typical month's printout. Any number of months can be run in succession. Each time you activate routine 2, register A returns to zero while register B continues to accumulate figures. Although this is called the New Month routine, any period of time may be used (week, month, quarter, etc.) Simply type the actual period involved when MONTH? appears.

DECEMBER 1981

ITEM	DATE	AMOUNT	CHECK #	CODE	PAID TO
1	12 1 81	6.75		34	SEATTLE TIMES
2	12 2 81	3.06		34	WIGWAM
3	12 2 81	1.36		3	SAFEWAY STORES
4	12 3 81	10.54	R330	34	JOHNSON TV
5	12 4 81	310	R331	1	THE HIGHLANDER APARTMENTS
6	12 7 81	18.43	R332	6	AMERICAN EXPRESS
7	17 7 81	67.16	R332	10	AMERICAN EXPRESS
8	12 7 81	111.69	R332	12	AMERICAN EXPRESS
9	12 4 81	6.32		3	SAFEWAY STORES
10	12 10 81	10.54	R333	34	JOHNSON TV
11	12 11 81	20.95	R334	3	SAFEWAY STORES
12	12 11 81	12.86	R335	17	J K GILL
13	12 11 81	15.79	R336	34	WHEREHOUSE
14	12 13 81	157.46	R337	31	G.M.A.C.
15	12 18 81	38.14	R338	3	SAFEWAY STORES
16	12 18 81	68.98	R339	2	PUGET POWER
17	12 23 81	20	R340	35	V.M.K.
18	12 25 81	32.28	R341	3	SAFEWAY STORES
19	12 26 81	10.54		34	JOHNSON TV
20	12 26 81	22.28	R342	5	BELL TELEPHONE
21	12 26 81	2.34		30	ERNST
22	12 31 81	84.26	R343	34	MAGNOLIA HI FI
23	12 31 81	310	R344	1	THE HIGHLANDER APARTMENTS
24	12 16 81	257		35	V.M.K.
25	1 23 81	257		35	V.M.K.
26	1 30 81	257		35	V.M.K.

```
DECEMBER 1981
0       $ 0
1       $ 620
2       $ 68.98
3       $ 99.05
4       $ 0
5       $ 22.28
6       $ 18.43
7       $ 0
8       $ 0
9       $ 0
10      $ 67.16
11      $ 0
12      $ 111.69
13      $ 0
14      $ 0
15      $ 0
16      $ 0
17      $ 12.86
18      $ 0
19      $ 0
20      $ 0
21      $ 0
22      $ 0
23      $ 0
24      $ 0
25      $ 0
26      $ 0
27      $ 0
28      $ 0
29      $ 0
30      $ 2.34
31      $ 157.46
32      $ 0
33      $ 0
34      $ 141.48
35      $ 791
36      $ 0
37      $ 0
38      $ 0
39      $ 0
```

Figure 1. *Monthly printout*

Summation to Date (routine 3)

After running for two or more months, you can access the cumulative totals to date from register B by running this routine. The computer first asks you for the period involved, then prints out a list of category numbers with

the amounts for each. Again, when the run is complete, control returns to the master menu.

Year End Summation (routine 4)

In this routine, the various categories are grouped into larger units and totaled. You can rearrange this segment of the program to suit your particular needs based on your category assignments.

Before executing the summary, the program asks for the following: ENTER ADJUSTED GROSS INCOME FROM FORM 1040 and ENTER DEPRECIATED VALUE. The adjusted gross income is used in the medical deduction feature of this routine. If you type in a zero, the program bypasses that feature; otherwise, the program works over the medical categories and produces a list of medical figures that can be copied onto your Form 1040. The program takes into account the one percent and three percent limits and the halved insurance entry for medical deductions in arriving at the medical figures it produces.

The depreciated value function is to be used when you have expenses for deductible equipment that must be handled on a depreciation schedule rather than as a straight deduction. (For instance, if a professional writer purchases a typewriter, this is a deductible expense, but the amount must be deducted a little each year.) The figure to be entered in response to the question is that amount to be deducted for the year for all depreciable equipment. You should refer to a good tax guide for instructions on depreciable deductions. The Year End Summation program then lists each major group, its categories and totals, and a group total. Figure 2 gives a typical Year End Summation. Upon completion of the printout, control returns to the master menu.

Flag Accounts (routine 5)

Occasionally, you have expenses that you want to track, but not to the extent of establishing a separate category. The flag account routine provides that tracking by giving a printed record of amounts paid out for certain products, services, etc. The numbers recorded by this routine do not change either register A or register B. You can use any number of flag accounts, and you can have a different set every time you run the program.

The routine asks for the period of time involved and the number of accounts to be set up in that run. It then requests the following for each account individually: the account title, the regular account that carried the expense, and the individual amounts to be entered. Any number of amounts may be entered for each account. When all the amounts have been entered, type a 0, and the program prints out the account title, the regular expense code, and the total for that account. Then the program moves on to the next flag account. This continues until you have entered

data for all the flag accounts. Control then returns to the master menu.

Register Value Correction (routine 6)

There are two ways to access this routine: from the master menu or from the New Month routine. The Register Value Correction routine does not automatically return you to the master menu; you determine whether control goes to the master menu or back to the New Month routine.

WORK AWAY FROM HOME EXPENSE	
LODGING	6421.52
UTILITIES	673.48
FOOD & MEALS	1884.31
TRANSPORTATION	491.4
TOTAL	9470.71
WORK & PROFESSIONAL EXPENSE	
WORK SEARCH	
MISC EXPENSE	34.8
PRINTED MATERIAL	69.6
TRANSPORTATION	69.6
LODGING	932.05
MEALS	124.96
LICENSES	715.28
PROF. PUBLICATIONS	160.16
PROF EQUIP & SUPPLIES	209.25
TOTAL	3184.06
WRITING EXPENSE	
POSTAGE	3.8
STATIONERY	974.72
BOOKS & PUBLICATIONS196.3	
FEES, AGENT ETC	150
TRAVEL	1145.76
COPIES, RESEARCH, TESTS ETC.	74.5
SPECIAL PROJECTS	1100
P.O. BOX RENT	71
TOTAL	3716.08
MEDICAL EXPENSE	
DOCTOR CALLS	400
MEDICINE	300
INSURANCE	2400
HOSPITAL COST	2000
MISC MEDICAL	75.9
TOTAL	5175.9

figure continued

SCHED A LINE NO	A M T $
1	1200
2	300
3	500.21
4	0
5	1200
6a	400
6b	2000
6c	75.9
7	3675.9
8	1500.63
9	2175.27
10	3375.27

ALIMONY	13626
CHARITY	24
DEPRECIABLE EQUIPMENT SEE ATTACHED DESCRIPTION	
DEPRECIATION	280
ENTERTAINMENT	3076.36
OTHER HOUSEHOLD EXPENSE	
CLOTHING	314.83
FURNISHINGS	2560
AUTO PAYMENTS	2046.98
AUTO LICENSE & INSUR	609.7
TOTAL	5531.51
UNACCOUNTED EXPENSES	455.66

Figure 2. *Year end summation*

To use this routine you must select the category to be corrected and enter the correct amount for both register A and register B. When these steps have been completed, you are given three courses of action. The computer asks for OTHER REGISTER CORRECTIONS and gives you three possible answers: Yes, Return to Master Menu, or Return to Current Month. You can correct any number of categories with this routine. If there are a number of categories to be corrected, however, it might be wiser and easier to end the run and reload register B on a new run by way of the Carry-over routine.

Notes and Instructions (routine 7)

This routine gives simplified notes and instruction for the program. If memory is limited, you can omit this section.

End Program (routine 8)

This routine ends the program execution.

Special Data Recapture (routine 9)

In the event the New Month routine is aborted before it is completed (due to power failure, fatal program error, deliberate shut off, etc.), it is possible to pick up where you left off by using the Special Data Recapture routine, eliminating the chore of reentering all the receipts. The first thing to do is to bring register B back to the point at which you started the aborted month by means of the Carry-over routine, then use routine 9 to reload registers A and B to the point at which the month's run was aborted. This routine is a scaled-down version of the New Month routine that requires only the amount and expense code. When the last item has been entered, type a zero, a comma, and a zero, and the control will move to the data entry of the New Month Routine and you can continue entering data for the aborted month.

Program Notes and Changes

As presented here, the program uses the following CHR$ codes:

CHR$(31)	Double-wide letters
CHR$(30)	Return from double-wide letters
CHR$(13)	Carriage return.

Since different printers use different codes for double-wide letters, a change in the program may be necessary to suit your printer. Most printers, however, will recognize the CHR$(13) code for a carriage return. If a 32-column printer (such as Radio Shack's Quick Printer II) is used, the program will have to be modified as shown in Table 3. The Program Listing gives a complete program list for an 80-column printer.

```
2070  LPRINT CHR$(15) C$ CHR$(13)"DATE";TAB(12);"CHECK";TAB(22);"AMT";
2080  LPRINT TAB(28);"CODE"
2430  LPRINT P;TAB(4);A$ CHR$(13) B$;TAB(12);D$;TAB(19);C;TAB(27);K
2460  LPRINT"*** ERROR IN LAST ENTRY ***"
11080 LPRINT N,TAB(4);"$";TAB(8);A(N);TAB(15);B(N)
      IN LINES 4030 THROUGH 4710 CHANGE:
            TAB(35)  TO  TAB(24)
            TAB(32)  TO  TAB(22)
```

Table 3. *Alternate list for 32-column Quick Printer II*

Program Listing. *Expense Account*

Encyclopedia Loader

```
1     REM **********  EXPENSE ACCOUNT PROGRAM  **********
2     REM IN THIS PROGRAM CHR$(13) IS USED AS A CARRIAGE RETURN
3     REM                 CHR$(31) IS USED TO LPRINT DOUBLE WIDE LETTER
      S
4     REM                 CHR$(30) RETURNS FROM DOUBLE WIDE LPRINTED LE
      TTERS
5     REM VARIABLES USED   C   GH JK MN  PQRST  W YZ
6     REM                A$ THROUGH G$
7     REM                AA THROUGH AH AND BA
8     REM                A(n) AND B(n)
10  CLEAR 1000
15    LET BA=39
20  DIM A(BA)
30  DIM B(BA)
31        FOR N=0 TO BA
32        LET B(N)=0
33        NEXT N
34    REM ABOVE SETS B REGISTER TO ZERO
35  CLS
40  PRINT:PRINT:PRINT
50    PRINT TAB(15);"      SELECT ROUTINE DESIRED"
51    PRINT TAB(16);"       1  =  CARRY-OVER INPUTS"
52    PRINT TAB(16);"       2  =  NEW MONTH INPUTS"
53    PRINT TAB(16);"       3  =  SUMMATION TO DATE"
54    PRINT TAB(16);"       4  =  YEAR END SUMMATION"
55    PRINT TAB(16);"       5  =  FLAG ACCOUNTS"
56    PRINT TAB(16);"       6  =  REGISTER VALUE CORRECTION"
57    PRINT TAB(16);"       7  =  NOTES AND INSTRUCTIONS"
58    PRINT TAB(16);"       8  =  END PROGRAM"
59    PRINT TAB(16);"       9  =  SPECIAL DATA RECAPTURE"
100    LET P=0
110   INPUT Z
115 IF Z>9 THEN 40
120    ON Z GOTO 1000,2000,3000,4000,5000,6000,9000,10000,11000
1000 REM *****  CARRY-OVER INPUT SUB-ROUTINE  *****
1010    FOR N=0 TO BA
1020 PRINT N
1030   INPUT Y
1040    LET B(N)=B(N)+Y
1050 NEXT N
1060 CLS
1070 GOTO 40
1080 STOP
2000 REM *****  NEW MONTH SUB-ROUTINE  *****
2010    FOR N=0 TO BA
2020    LET A(N)=0
2030 NEXT N
2031 REM ******** ABOVE RETURNS REGISTER A TO ZERO ********
2040 CLS
2050 PRINT"MONTH ???"
2060   INPUT C$
2070 LPRINTCHR$(31) C$ CHR$(30)CHR$(13)"ITEM";TAB(10);"DATE";TAB(19);"
     AMOUNT";
2080 LPRINT TAB(28);"CHECK #";TAB(39);"CODE";TAB(46);"PAID TO"
2090 PRINT"PAID TO, CHECK NUMBER"
2091 PRINT"                          END = END OF MONTH"
2092 PRINT"                          ERROR GOES TO REGISTER CORRECTION ROU
     TINE"
2100   INPUT A$,D$
2101 REM **** LINES 2110 THROUGH 2360 RESERVED FOR A$ CODES
2110    IF A$="A" THEN A$="AMERICAN EXPRESS"
2120    IF A$="B" THEN A$="BELL TELEPHONE"
2130    IF A$="BK" THEN A$="BAKER TRAILS CO."
2140    IF A$="C" THEN A$="COLUMBIA HOUSE"
2150    IF A$="CH" THEN A$="CACHET HOMES CO."
2160    IF A$="D" THEN A$="DIXIE ELECTRIC POWER CO-OP"
2170    IF A$="G" THEN A$="G.M.A.C."
```

```
2180   IF A$="GS" THEN A$="GULF STATES ELECTRIC UTILITIES"
2190   IF A$="H" THEN A$="THE HIGHLANDER APARTMENTS"
2260   IF A$="P" THEN A$="PUGET POWER"
2280   IF A$="R" THEN A$="RADIO SHACK"
2281   IF A$="RCA" THEN A$="RCA MUSIC SERVICE"
2290   IF A$="S" THEN A$="SAFEWAY STORES"
2340   IF A$="V" THEN A$="V.M.K."
2350   IF A$="VL" THEN A$="VIDIO LIBRARY"
2370   IF A$="END" THEN 2500
2375   IF A$="ERROR" THEN 2460
2380   LET P=P+1
2390 PRINT"DATE, AMOUNT, EXPENSE CODE"
2400  INPUT B$,C,K
2410   LET A(K)=A(K)+C
2420   LET B(K)=B(K)+C
2430 LPRINT P;TAB(8);B$;TAB(19);C;TAB(30);D$;TAB(39);K;TAB(46);A$
2440 GOTO 2090
2450 STOP
2460 LPRINT"*** ERROR IN ITEM ";P;" DELETE ENTRY ***"
2465   LET P=P-1
2470 GOSUB 6000
2480 GOTO 2090
2490 STOP
2500 LPRINT:LPRINT TAB(5);"TOTALS FOR THE MONTH OF ";C$;CHR$(13)CHR$(1
     3);TAB(6);"CATEGORY","AMOUNT"
2510   FOR N=0 TO BA
2520 LPRINT:LPRINT TAB(10);N,"$ ";A(N)
2530 NEXT N
2540 CLS
2550 GOTO 40
2560 STOP
3000 REM ***** SUMMATION TO DATE SUB-ROUTINE *****
3010 PRINT"CUMULATIVE TOTALS THROUGH ???"
3020   INPUT E$
3030 LPRINT"CUMUMATIVE TOTALS THROUGH ";E$
3040   FOR N=0 TO BA
3050 LPRINT N,"$ ";B(N)
3060 NEXT N
3070 CLS
3080 GOTO 40
3090 STOP
3900 REM *****_YEAR END SUB-ROUTINE *****
4000 PRINT"ENTER ADJUSTED GROSS INCOME FROM FORM 1040"
4005  INPUT W
4010 PRINT"ENTER DEPRECIATED VALUE"
4015  INPUT H
4020 LPRINT
4030 LPRINT"WORK AWAY FROM HOME EXPENSE"
4040 LPRINT
4050 LPRINT"    LODGING"TAB(35);B(1)
4060 LPRINT"    UTILITIES"TAB(35);B(2)
4070 LPRINT"    FOOD & MEALS";TAB(35);B(3)
4080 LPRINT"    TRANSPORTATION"TAB(35);B(29)+B(30)
4090 LPRINT
4100 LPRINT"TOTAL";TAB(32);B(1)+B(2)+B(3)+B(29)+B(30)
4110 LPRINT
4120 LPRINT"WORK & PROFESSIONAL EXPENSE"
4130 LPRINT
4140 LPRINT"    WORK SEARCH"
4150 LPRINT"      MISC EXPENSE"TAB(35);B(8)
4160 LPRINT"      PRINTED MATERIAL";TAB(35);B(9)
4170 LPRINT"      TRANSPORTATION";TAB(35);B(10)
4180 LPRINT"      LODGING";TAB(35);B(12)
4190 LPRINT"      MEALS";TAB(35);B(11)
4200 LPRINT"    LICENSES"; TAB(35);B(13)
4210 LPRINT"    PROF. PUBLICATIONS";TAB(35);B(14)
4220 LPRINT"    PROF. EQUIP & SUPPLIES";TAB(35);B(15)
4230 LPRINT
4240 LPRINT"TOTAL";TAB(32);B(8)+B(9)+B(10)+B(11)+B(12)+B(13)+B(14)+B(1
     5)
```

Program continued

```
4250 LPRINT
4260 LPRINT"WRITING EXPENSE"
4270 LPRINT
4280 LPRINT"    POSTAGE";TAB(35);B(16)
4290 LPRINT"    STATIONERY";TAB(35);B(17)+B(18)
4300 LPRINT"    BOOKS & PUBLICATIONS";TAB(35);B(19)
4310 LPRINT"    FEES, AGENT ETC";TAB(35);B(20)
4320 LPRINT"    TRAVEL";TAB(35);B(21)
4330 LPRINT"    COPIES, RESEARCH, TESTS ETC.";TAB(35);B(22)
4340 LPRINT"    SPECIAL PROJECTS";TAB(35);B(39)
4350 LPRINT"    P.O. BOX RENT";TAB(35);B(23)
4360 LPRINT
4370 LPRINT"TOTAL";TAB(32);B(16)+B(17)+B(18)+B(19)+B(20)+B(21)+B(22)+B
     (23)+B(39)
4380 LPRINT
4390 LPRINT"MEDICAL EXPENSE"
4400 LPRINT
4410 LPRINT"    DOCTOR CALLS";TAB(35);B(27)
4420 LPRINT"    MEDICINE";TAB(35);B(26)
4430 LPRINT"    INSURANCE";TAB(35);B(25)
4435 LPRINT"    HOSPITAL COST";TAB(35);B(24)
4440 LPRINT"    MISC MEDICAL";TAB(35);B(28)
4450 LPRINT
4460 LPRINT"TOTAL";TAB(32);B(24)+B(25)+B(26)+B(27)+B(28)
4470 IF W>0 GOSUB 4800
4480 LPRINT
4490 LPRINT"ALIMONY";TAB(32);B(35)
4500 LPRINT
4510 LPRINT"CHARITY";TAB(32);B(36)
4520 LPRINT
4530 LPRINT"DEPRECIABLE EQUIPMENT"
4540 LPRINT"  SEE ATTACHED DESCRIPTION"
4570 LPRINT
4580 LPRINT"DEPRECIATION";TAB(32);H
4590 LPRINT
4600 LPRINT"ENTERTAINMENT";TAB(32);B(34)
4610 LPRINT
4620 LPRINT"OTHER HOUSEHOLD EXPENSE"
4630 LPRINT
4640 LPRINT"    CLOTHING";TAB(35);B(6)
4650 LPRINT"    FURNISHINGS";TAB(35);B(7)
4660 LPRINT"    AUTO PAYMENTS"TAB(35);B(31)
4670 LPRINT"    AUTO LICENCE & INSUR";TAB(35);B(32)
4680 LPRINT
4690 LPRINT"TOTAL";TAB(32);B(6)+B(7)+B(31)+B(32)
4700 LPRINT
4710 LPRINT"UNACCOUNTED EXPENSES";TAB(35);B(0)
4720 LPRINT
4730 CLS
4740 GOTO 40
4750 STOP
4800 REM *****TAX FORM MEDICAL CALCULATIONS *****
4810 LPRINT
4820   LET AB=B(25)/2
4821   LET AC=B(26)
4822   LET AD=W*.01
4823   LET AE=AC-AD
4824    IF AE=<0 THEN AE=0
4825   LET AF=AE+AB+B(27)+B(24)+B(28)
4826   LET AG=W*.03
4827   LET AH=AF-AG
4828    IF AG=>AF THEN AH=0
4840 LPRINT"SCHED A     AMT"
4850 LPRINT"LINE NO      $"
4860 LPRINT
4870 LPRINT"  1";TAB(14);AB
4880 LPRINT"  2";TAB(14);AC
4890 LPRINT"  3";TAB(14);AD
4900 LPRINT"  4";TAB(14);AE
4910 LPRINT"  5";TAB(14);AB
```

```
4920 LPRINT"  6a";TAB(14);B(27)
4930 LPRINT"  6b";TAB(14);B(24)
4940 LPRINT"  6c";TAB(14);B(28)
4950 LPRINT"  7";TAB(14);AF
4960 LPRINT"  8";TAB(14);AG
4970 LPRINT"  9";TAB(14);AH
4980 LPRINT" 10";TAB(14);AB+AH
4990 RETURN
5000 REM *****FLAG ACCOUNT ROUTINE *****
5050 PRINT"INPUT MONTH OR PERIOD COVERED"
5060   INPUT G$
5070 LPRINT CHR$(13)CHR$(31)"FLAG ACCOUNTS"CHR$(30)CHR$(13)"   FOR
    "G$
5100 PRINT"HOW MANY FLAG ACCOUNTS???"
5110  INPUT G
5120   FOR N=1TOG
5130 PRINT"ENTER TITLE OF ACCOUNT ";N;" AND REGULAR EXPENSE CODE"
5140  INPUT F$,H
5150   LET J=0
5160 PRINT"AMOUNTS TO BE INCLUDED IN FLAG ACCOUNT ";N
5170 PRINT"TYPE 0 FOR LAST ENTRY"
5180 INPUT M
5190   IF M=0 THEN 5210
5195   LET J=J+M
5200 GOTO 5180
5210 LPRINT F$
5220 LPRINT"PART OF REGULAR ACCOUNT ";H
5230 LPRINT"    $  ";J
5240 LPRINT
5300 NEXT N
5310 CLS
5320 GOTO 40
5330 STOP
6000 REM *****REGISTER CORRECTION SUB-ROUTINE *****
6010 PRINT"TYPE IN NUMBER OF CATEGORY TO BE CORRECTED"
6020  INPUT Q
6030 PRINT
6040 PRINT"A", "B"
6050 PRINT
6060 PRINT A(Q),B(Q)
6070 PRINT
6080 PRINT"TYPE IN CORRECT A & B REGISTER VALUES"
6090  INPUT R,S
6100 PRINT
6110   LET A(Q)=R
6120   LET B(Q)=S
6130 PRINT"NEW REGISTER VALUES"
6140 PRINT A(Q),B(Q)
6150 PRINT
6160 PRINT"OTHER REGISTER CORRECTIONS???"
6161 PRINT"     1 = YES"
6162 PRINT"     2 = RETURN TO MASTER MENU"
6163 PRINT"     3 = RETURN TO CURRENT MONTH"
6170  INPUT T
6180   ON T GOTO 6000,40,6200
6190 STOP
6200 RETURN
6210 STOP
9000 REM ***** NOTES AND INSTRUCTIONS *****
9010 CLS
9040 PRINT"PROGRAM MAINTAINS TWO REGISTERS FOR CATEGORIZING EXPENSES"
9050 PRINT TAB(17);"REGISTER A = MONTHLY TOTALS. RETURNS"
9060 PRINT TAB(30);"TO 0 WITH EACH NEW MONTH"
9080 PRINT
9090 PRINT TAB(17);"REGISTER B = CUMULATIVE TOTALS, TOTALIZES"
9100 PRINT TAB(30);"ACCTS ENTERED VIA CARRY-OVER"
9110 PRINT TAB(30);"AND MONTHLY ROUTINES"
9130 PRINT
9140 PRINT"   REGISTER A PRINTS OUT AT THE END OF EACH MONTHS RUN"
9150 PRINT"     THEN RETURNS TO ZERO"
```

Program continued

```
9170 PRINT"   REGISTER B PRINTS OUT ONLY VIA SUMMATION TO DATE"
9180 PRINT"     ROUTINE"
9190 PRINT"            TYPE 1 TO CONTINUE OR 0 TO RETURN TO MENU"
9200  INPUT AA
9210   IF AA=0 THEN 40
9220 CLS
9230 PRINT"CARRY-OVER INPUTS   **USE TO INPUT DATA FROM PREVIOUS PERIO
     DS"
9240 PRINT TAB(20);"MAY BE REPEATED AS MANY TIMES AS REQUIRED"
9260 PRINT"
9270 PRINT"NEW MONTH INPUTS    **USE TO INPUT MONTHLY EXPENSES. THIS"
9280 PRINT TAB(20);"ROUTINE TOTALIZES EXPENSES BY CATEGORY"
9290 PRINT TAB(20);"FOR THE MONTH"
9300 PRINT
9310 PRINT"  NOTE: TYPING   END   WILL END MONTHLY INPUTS"
9320 PRINT TAB(17);"ERROR  WILL SHIFT TO REGISTER CORRECTION"
9330 PRINT TAB(17);"ROUTINE (SEE BELOW)"
9340 PRINT
9350 PRINT"             TYPE 1 TO CONTINUE OR 0 TO RETURN TO MENU"
9360  INPUT AA
9370   IF AA=0 THEN 40
9380 CLS
9420 PRINT"SUMMATION TO DATE  **PROVIDES A PRINTOUT OF REGISTER B"
9430 PRINT
9440 PRINT"YEAR END SUMMATION **PROVIDES A YEAR END TOTALIZATION BY MA
     JOR"
9450 PRINT TAB(20);"AREAS WHICH MAY INCLUDE SEVERAL CATEGORIES"
9470 PRINT
9480 PRINT"FLAG ACCOUNTS       **PROVIDES A METHOD OF TRACKING"
9490 PRINT TAB(20);"SPECIFIC PORTIONS OF AN EXISTING ACCT."
9500 PRINT TAB(20);"THIS ROUTINE DOES NOT AFFECT REGISTERS"
9510 PRINT TAB(20);"A OR B, BUT DOES PROVIDE A PRINTED"
9520 PRINT TAB(20);"RECORD OF THE EXPENSE INVOLVED"
9530 PRINT
9540 PRINT
9550 PRINT"        TYPE 1 TO CONTINUE OR 0 TO RETURN TO MENU"
9560  INPUT AA
9570   IF AA=0 THEN 40
9580 CLS
9610 PRINT"REGISTER VALUE     **PROVIDES A MEANS OF CORRECTING VALUES"

9620 PRINT"  CORRECTION        IN BOTH REGISTERS FOR ANY EXPENSE CATEG
     ORY"
9640 PRINT
9650 PRINT TAB(20);"THIS ROUTINE MAY BE ADDRESSED FROM"
9660 PRINT TAB(20);"THE MASTER MENU OR FROM THE MONTHLY"
9670 PRINT TAB(20);"INPUT ROUTINE.  FROM THE MONTHLY ROUTINE"
9680 PRINT TAB(20);"TYPE    ERROR,   IN PLACE OF PAY TO."
9690 PRINT
9700 PRINT TAB(20);"WHEN USING THIS ROUTINE FROM THE MONTHLY"
9710 PRINT TAB(20);"ROUTINE, BE SURE TO TYPE THE PROPER"
9720 PRINT TAB(20);"RETURN TO CODE TO GO BACK TO THE MONTHLY"
9730 PRINT TAB(20);"ROUTINE, OTHERWISE THE MONTH IN PROGRESS"
9740 PRINT TAB(20);"WILL BE ABORTED AND B REGISTER WILL NOT"
9750 PRINT TAB(20);"BE CORRECT"
9760 PRINT"       TYPE 1 TO CONTINUE OR 0 TO RETURN TO MENU"
9770  INPUT AA
9780   IF AA=0 THEN 40
9790 CLS
9800 PRINT"SPECIAL DATA       **PROVIDES A MEANS OF RELOADING REGISTER
     "
9810 PRINT "  RECAPTURE         A IN THE EVENT OF A FATAL PROGRAM ERRO
     R"
9820 PRINT TAB(20);"OR POWER FAILURE OR OTHER INTERRUPTION"
9830 PRINT TAB(20);"THIS IS A SCALED DOWN VERSION OF THE NEW"
9840 PRINT TAB(20);"MONTH ROUTINE WHERE ONLY THE AMOUNT AND"
9850 PRINT TAB(20);"EXPENSE CODE ARE INPUT.  USE CARRY-OVER"
9860 PRINT TAB(20);"FIRST TO BRING REGISTER B UP TO DATE"
9870 PRINT TAB(20);"THEN THIS ROUTINE.  THIS ROUTINE WILL"
9880 PRINT TAB(20);"RETURN TO THE DATA INPUT STAGE OF THE"
```

```
9890 PRINT TAB(20);"NEW MONTH ROUTINE, THUS IT IS POSSIBLE"
9900 PRINT TAB(20);"TO CARRY ON WITH AN ABORTED MONTH"
9910 PRINT"       TYPE 1 TO CONTINUE"
9920    INPUT AA
9930 CLS
9940 GOTO 40
10000 END
11000 REM    ***** SPECIAL RECAPTURE SUB-ROUTINE *****
11010 PRINT"ENTER AMOUNT AND CODE"
11020 INPUT C,K
11025 IF C=0 AND K=0 THEN 11070
11030 LET A(K)=A(K)+C
11040 LET B(K)=B(K)+C
11050 PRINT C,,K
11060 GOTO11010
11070   FOR N=0 TO BA
11080 LPRINT N,"$ ";A(N);"
11090 NEXT N
11100 CLS
11110 GOTO 2040
```

INTERFACE

Model III I/O Port

INTERFACE

Model III I/O Port

by Harry Avant

Several months ago, I constructed an interface for a Model I that received signals from two photographic densitometers and several temperature indicators. That I/O device was memory mapped starting at 3000 hex. A recent acquisition of a Model III required a new I/O interface for a similar application. This article describes some of the characteristics of the Model III I/O bus and the basic interface, but none of the instruments attached to it.

The external bus for the Model III is somewhat different than the one used by a Model I. Table 1 gives a description of the Model III bus. An asterisk indicates that a logic level zero is true.

Radio Shack publishes a very good service manual for the Model III, *TRS-80 Model III Service Manual*, part number 26-1061/1062/1062. This publication should be acquired by anyone designing or constructing add-on devices to the Model III.

On the Model III, only the lower eight address lines are brought out, restricting any interface to port-based I/O only. Memory mapped I/O is not possible due to the absence of the upper address lines. Not all of the possible 256 I/O ports are available for your use because the Model III requires ports 80H through FFH for internal uses, such as the line printer address and disk drives. There are still more than 100 port addresses you can use.

The Model III bus is protected by buffers that are enabled in the outgoing direction by software. Incoming signals on the data bus must have an external signal supplied in order to enable the buffers for incoming data.

Figure 1 is a diagram of the bus buffering of a Model III. The even numbered pins are all ground and are located on the component side of the printed circuit board. The signal ENEXTIO is generated by software at port 128 decimal to enable U101–103. Until activated by a signal from port 128, the edge card J2 is disconnected. Pin 43 of J2 is another change from the Model I. This line, EXTIOSEL*, must go low in order to allow incoming signals on the data lines to pass through U101. When using this line, note that a 150 Ohm resistor is tied between the line and Vcc. This requires that the chip used to drive the line be capable of handling the resultant load. A very useful control line, IOBUSINT*, is on pin 39 of J2. This is a Z-80 mode 1 interrupt which, when pulled low by an external device, forces the computer to jump to the service routine whose address is stored at 403EH and 403FH. The Model III external bus has a much better layout than that of the

Number	Function
1	Data line 0
3	Data line 1
5	Data line 2
7	Data line 3
9	Data line 4
11	Data line 5
13	Data line 6
15	Data line 7
17	Address A0
19	Address A1
21	Address A2
23	Address A3
25	Address A4
27	Address A5
29	Address A6
31	Address A7
33	IN* specifying that an input is in progress
35	OUT* specifying that an output is in progress
37	RESET* system reset
39	IOBUSINT* notifies the CPU that an interrupt is requested
41	IOBUSWAIT* to force wait states from external devices
43	EXTIOSEL* enables input via bi-directional data buffer
45	No connection
47	MI* standard Z-80 signal
49	IORQ* standard Z-80 signal
2–50	All even numbered pins are ground.

Table 1. *Description of the Model III bus*

Model I, with all even numbered pins at ground and data and address lines in numerical order.

The Intel 8255 programmable interface chip used in this interface has been described several times in articles relating to its use in the TRS-80 Model I. An extensive discussion of the programming of the 8255 is not included here. A brief example of the use of control words in mode 0 is shown in Figure 2.

A major difference in using the 8255 with the Model III is the capability of incorporating interrupt mode 1 applications. In this mode, the 8255 can directly cause the TRS-80 to jump to the interrupt routine. Intel publishes an excellent data sheet for this device, Application Note AP-15, *8255A Programmable Peripheral Interface Applications*, which sells for $1. It

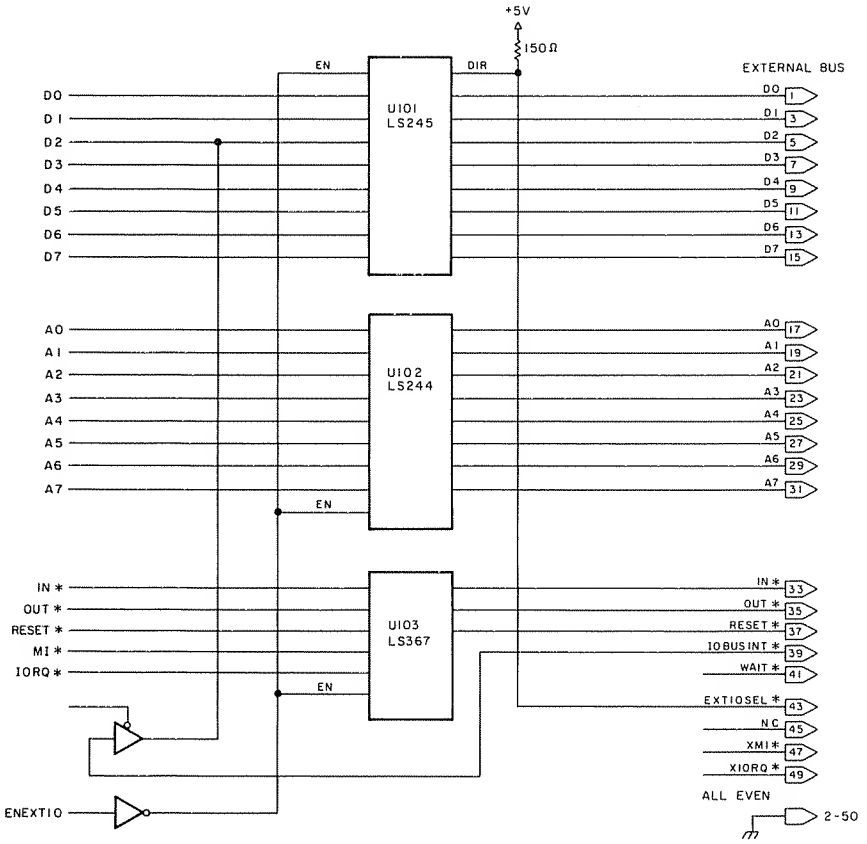


Figure 1. *Model III bus buffering*

describes several examples of interfacing the 8255, including the use of it with interrupts.

Figure 3 is a schematic of the basic 24-bit I/O device. A photograph of the basic interface is shown in Photo 1. Due to a unique physical placement problem for the interface described (located at the end of a six-foot, fifty-pin cable) some unusual precautions against noise have been employed. Photos 2 and 3 show ringing at both the leading and trailing edge of a data line at the end of a six-foot cable. For many applications, this amount of over- and undershoot would be catastrophic. Extending the data and address lines out over a six-foot cable can produce an extensive amount of radio frequency interference that can raise havoc with nearby television receivers and other high-frequency detectors.

In this circuit, Vcc pins for all of the integrated circuits are not connected directly to 5 volts, but are routed through 2.7 Ohm, 1/8 Watt resistors. This

A1	A0	RD*	WR*	CS*	Action
0	0	0	1	0	input port A to data bus.
0	1	0	1	0	input port B to data bus.
1	0	0	1	0	input port C to data bus.
0	0	1	0	0	output data bus to port A.
0	1	1	0	0	output data bus to port B.
1	0	1	0	0	output data bus to port C.
1	1	1	0	0	output control word to 8255.
X	X	X	X	1	tri-state
X	X	1	1	0	tri-state

Example: Control word sent to port 15 (OUT 15, control word)

Control word	Port A	Port B	Port C low	Port C upper
128	OUT	OUT	OUT	OUT
129	OUT	OUT	IN	OUT
130	OUT	IN	OUT	OUT
131	OUT	IN	IN	OUT
136	OUT	OUT	OUT	IN
137	OUT	OUT	IN	IN
138	OUT	IN	OUT	IN
139	OUT	IN	IN	IN
144	IN	OUT	OUT	OUT
145	IN	OUT	IN	OUT
146	IN	IN	OUT	OUT
147	IN	IN	IN	OUT
152	IN	OUT	OUT	IN
152	IN	OUT	IN	IN
153	IN	OUT	IN	IN
154	IN	IN	OUT	IN
155	IN	IN	IN	IN

Figure 2. *8255 configuration*

provides a fuse for each chip and, combined with the bypass capacitors at each chip, provides excellent decoupling.

The pull-up resistors used here are 1000 Ohms. Some Model I interfaces use pull-up resistors of 4700 to 10,000 Ohms. This is far too high a value in this application and can significantly reduce switching speed. In addition, pin 2 of U4 has the unused pin pulled up via a 1000 Ohm resistor in series with a 100 uf capacitor. I highly recommend this technique for pulling up unused gates on multiple input chips for noise suppression. Little additional features such as these are well worth the slight additional cost, as they produce clean switching and prevent burning out an entire board full of chips in the event of misapplied voltage.

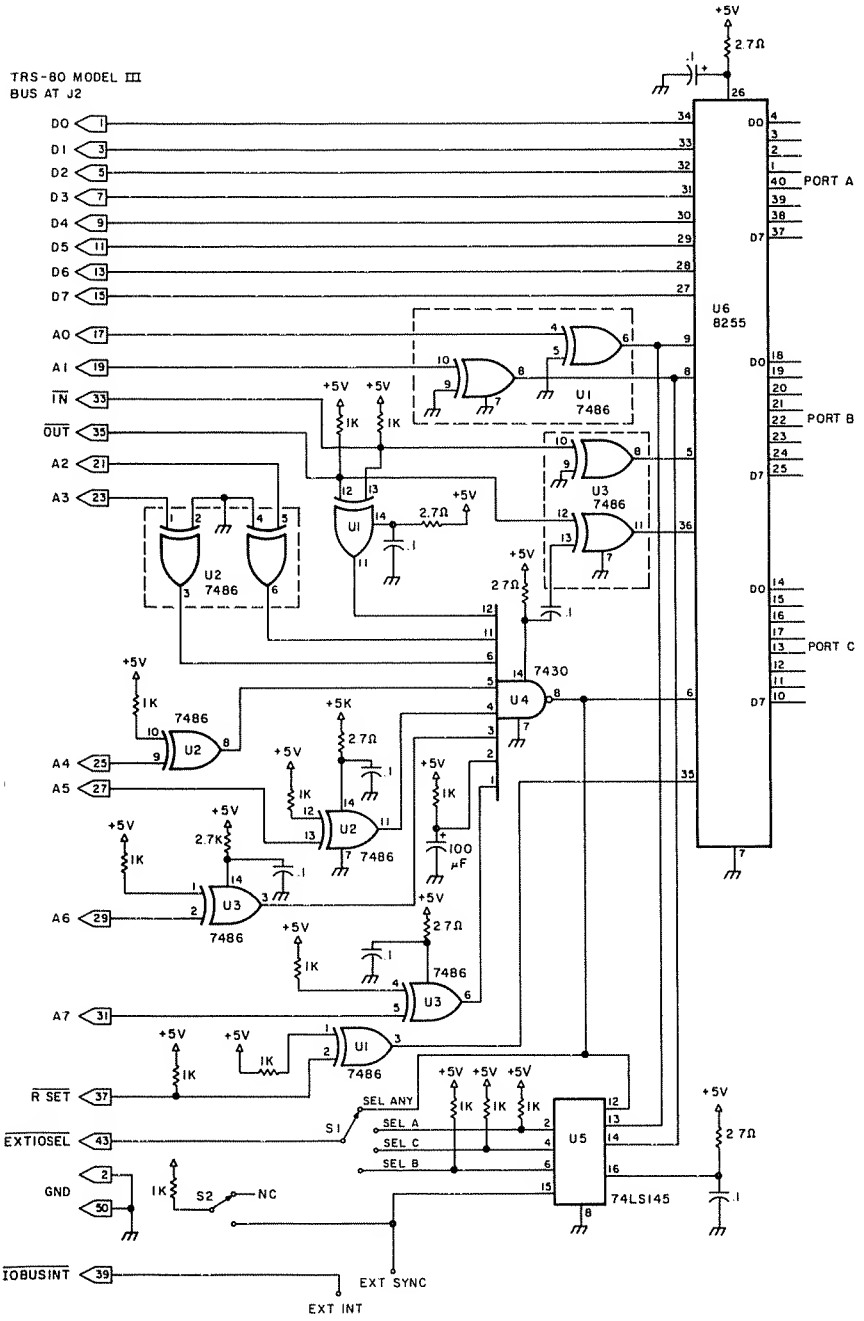


Figure 3. *24-bit I/O device*

Integrated circuits U1–U4 provide a conventional port-based decoder for addressing and writing to the 8255. Address lines A0,A1,A2, and A3 are

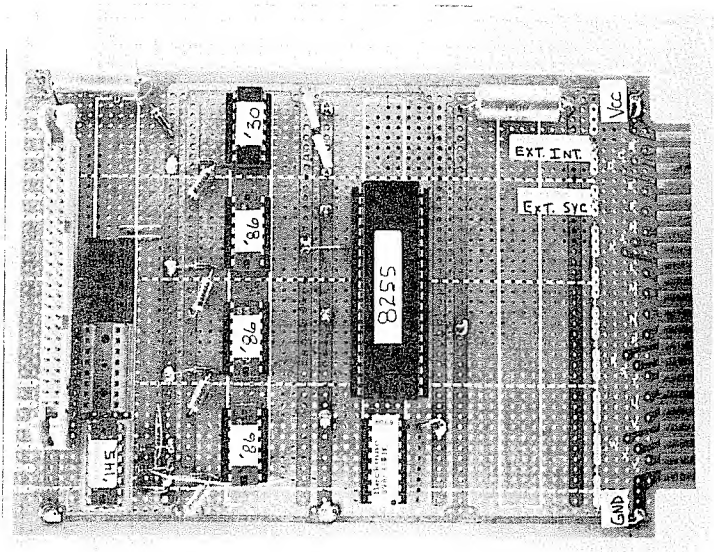


Photo 1

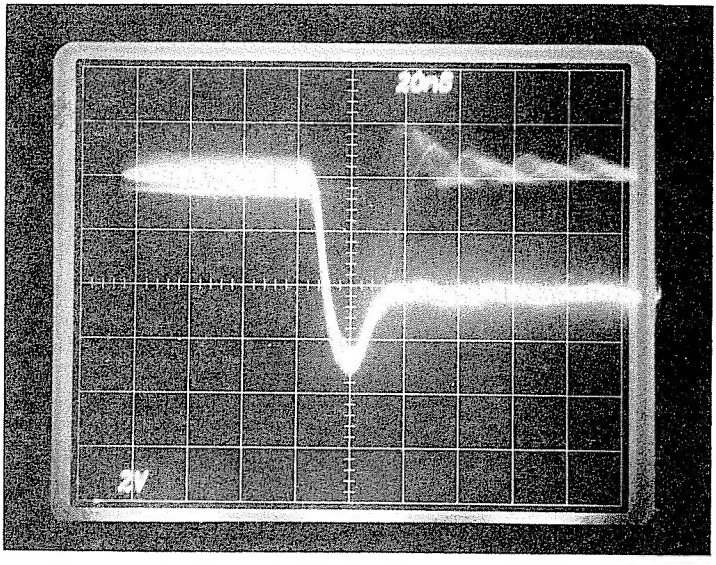


Photo 2

noninverted by U1 and U2, providing for a range of address from 12 to 15. The 8255 control word register is accessed by setting a bit high on both A0 and A1 so that port 15 is the control word address for the I/O chip. Address 12 represents port A, 13 represents port B, and 14 represents port C. When addressed, a pulse of about 1200 nanoseconds is sent to pin 6 of the 8255.

A 74LS145 chip provides synchronized input signals. I selected this chip because of the need to drive large loads that are not shown on the basic interface drawing. By setting the switch S-1, you can synchronize input with all ports or just one. In addition, switch S-2 provides an external sync signal. An additional line, EXTINT, is brought out to allow for external driving of the interrupt mode 1 capability.

After the interface has been assembled, you can test it using the simple BASIC program that follows. Use jumper wires to connect port A to port C, pin 4 to pin 18, pin 3 to pin 19, and so on.

```
10 OUT 236,16 : REM enable the I/O bus
20 OUT 15,137 : REM control word to configure the 8255
21 REM : for output port A and B input port C
30 FOR X = 1 TO 255
40 OUT 12,X : REM send out "X" to port A
50 C = INP(14) : REM read value coming into port C
52 PRINT"OUTPUT TO PORT A = "; X
54 PRINT"INPUT FROM PORT C = ";C
60 FOR T = 1 TO 10 : NEXT T
```

Program continued

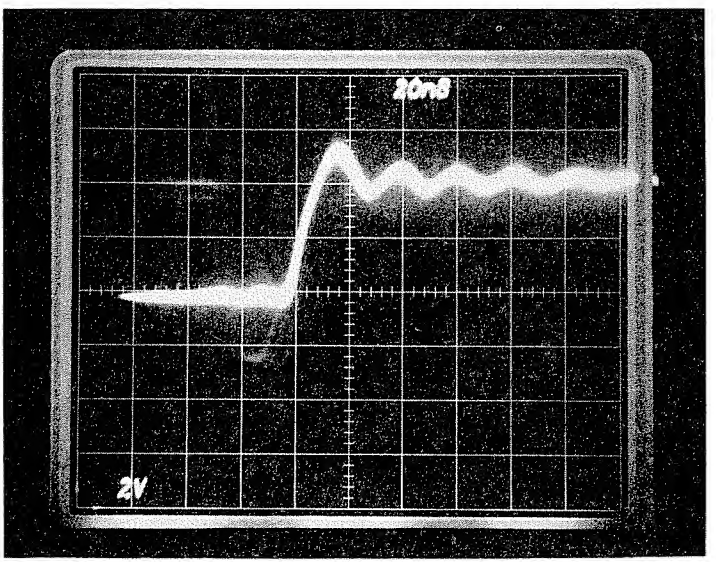


Photo 3

```
70 NEXT X
80 OUT 236,0 : REM turn off the I/O bus
90 GOTO 10
```

To test the interrupt mode 1 function, use the following program. Load the assembly-language code shown in Figure 4 into high memory first. Next, enter BASIC, reserve memory (65279), enter the BASIC code, and run the program. During the run, a momentary short from 39 to ground causes the interrupt function to occur, followed by a return to the BASIC program.

A typical BASIC program to test the interrupt action would be as follows:

```
100 POKE &H403E,&H00 : REM address of interrupt program
110 POKE &H403F,&HFF
120 OUT 236,16 : REM enable the external bus
130 OUT 224,8  : REM set bit 3 of port OE.
140 FOR R = 1 TO 10
150 FOR X = 33 TO 127
160 PRINT CHR$(X);
170 NEXT X
180 NEXT R
200 END
```

```
00240    ORG    0FF00H          ; for 48K,decimal 65280
00250 ;save the registers
00260    PUSH   AF
00270    PUSH   BC
00280    PUSH   DE
00290    PUSH   HL
00300    PUSH   IX
00310    PUSH   IY
00320 ;
00330    CALL   01C9H           ;Clear the video
00340    LD     HL,MSG1
00350    CALL   021BH           ;$VDLINE
00360 ;
00370    POP    IY
00380    POP    IX
00390    POP    HL
00400    POP    DE
00410    POP    BC
00420    POP    AF
00430 ;
00440    EI
00450 ;
00460    RETI
00465 ;
00470 MSG1     DEFM   'THIS IS A TEST OF IM-1'
00480    DEFB   0DH
```

```
00490 ;
00500    END     0FF00H
```

Figure 4. *Interrupt routine to be stored in high memory*

When these programs are run, you can enable the interrupt at any point during the execution of the BASIC program. The screen clears, and THIS IS A TEST OF IM-1 appears for a moment. Then the BASIC program resumes its execution without losing any characters.

TUTORIAL

A Bit of Precision
Computer Number Systems
And Arithmetic Operations—Part II

A Bit of Precision

by Allan S. Joffe W3KBM

There are many reasons why your computer may not do what you want it to do. In this article, I will describe some of these problems so you can examine their causes.

Perhaps the easiest way to begin is to use the SGN function in a program. Remember that SGN(X) returns −1 for values of X less than 0; it returns 0 if X = 0; it returns +1 if X is greater than 0.

Type in this programming segment for the initial successful demonstration.

```
 5 CLS
10 FOR X = -2 to 2 STEP .5
20 IF SGN(X) = 0 THEN STOP
30 NEXT X
```

When you run the program, you get a BREAK IN 20 message. To help you visualize what is going on, add:

```
15 PRINT X;....
```

This gives you the following series on your video screen:

```
-2  -1.5  -1  -.5  0
```

With SGN(X) equal to 0, the program stops.

To guarantee a complete lack of success, change line 10 to read:

```
10 FOR X = -2 TO 2 STEP .1
```

Before you run the revised program, think a bit about why success might elude you. If you play computer with pencil and paper, you will perform the indicated operations and reach 0. It appears that SGN(X) could equal 0, halting the program. Run the revision, including line 15, and you have successfully avoided success. To see why, look at the numbers on your screen. In the center of the second line of the printout you see this series of numbers.

```
-.2  -.0999997  3.129241 -07  .1
```

SGN(X) apparently has failed because, for some reason, the computer has not generated the zero that SGN(X) was looking for. The reason is that your computer cannot give you a fine-tuned value for .1. If you try step values of .25, or .0625, or .03125, then SGN(X) works once more. The logical answer lies in how your computer internally treats values to the right of the decimal point. If they are equal to 1/2, 1/4, 1/8, 1/16, and so on, SGN(X) works like a charm, because this series is the binary reciprocal of the series to the left of

the decimal point, namely, 2, 4, 8, 16, and so on. Now, let us throw a log or two into the hopper and get into some similar trouble with logarithms.

The charm of logs is that they can make the handling of large numbers simple. If you ask the computer to print the log of 2, it gives you an answer of .693147 which is indeed the log of 2 but which is more precisely the Naperian log of 2. You probably wanted the log of 2 to the base 10 or what is called the common log of 2. To get this answer, enter:

```
PRINT LOG(2)/LOG(10)
```

This gives you .30103, which is the value you probably wanted and expected to see.

At this point, it may seem illogical to present a log example for a small number when I suggested that the charm of logs was their facility with large numbers. Look at the following program.

```
 5 CLS
10 FOR X = 50000 TO 50005
20 PRINT LOG(X)/LOG(10);
30 NEXT X
```

The problem becomes painfully obvious when you run this program and see on the screen:

```
4.69897  4.69898  4.69899  4.699  4.69901  4.69901
```

It does not make sense to believe that 50004 and 50005 have identical values for their logs. The answer lies in insufficient precision at this level. I realize that the temptation is strong to add a line:

```
7 DEFDBL X
```

With this approach, however, you get a TM (type mismatch) error message.

The computer syntax evidently does not allow you to set the FOR/NEXT loop counter value to be a double-precision variable. To get around the problem, do a bit of logical program restructuring.

```
 5 CLS
 7 DEFDBL A
10 FOR X = 50000 TO 50005
20 A = LOG(X)/LOG(10)
30 PRINT A;
40 NEXT X
```

If you run this and examine the last two answers in the printout, you see that:

```
Log of 50004 = 4.699004650115967
Log of 50005 = 4.699013710021973
```

The double-precision feature allows you to get separation of these two values—separation that single precision would not allow.

For the values 50000 to 50005, you find that you really do not need to use more than seven of the digits to the right of the decimal point to implement an anti-log expression that lets you use the log of X to recover X accurately.

If you have the log of a number, you can find the number by exponentiation, that is, raising the base 10 of the common log system to a power which is the logarithm of the number. For example, you have the log of 50004. If you enter:

```
PRINT 10↑ 4.6990046
```

you would see 5004 on the screen.

Now you will put the computer through the wringer by asking it to work on a value in the millions.

```
 5 CLS
10 DEFDBL A,B
20 FOR X = 1000000 TO 1000010
30 A = LOG(X)/LOG(10)
40 B = 10↑A
50 PRINT A,: PRINT FIX(B)
60 NEXT X
```

This program first takes the log and then takes the anti-log of the values of X. FIX gets rid of any decimal values since they are totally meaningless. The anti-logs produced are as follows:

```
1000000
1000001
1000002
1000003
1000003
1000004
1000005
1000007
1000007
1000008
1000011
```

From the fact that 17 digits of precision are not enough to provide finite separation of the values on an anti-log basis, you might jump to the conclusion that the results have questionable value. But, if you consider the relative error of one or two parts out of 1,000,000, you can appreciate the value of the double precision that is packed into your computer.

For the sake of relative completeness, try one small change in the program. Change the second half of line 50 to read:

```
PRINT USING"#######";B
```

The anti-log values are as follows:

```
1000000
1000002
1000003
1000003
1000005
1000006
```

1000008
1000009
1000012

The slight changes noted here are due to the rounding off that is part of the PRINT USING statement.

Computer Number Systems And Arithmetic Operations—Part II

by Gene Kovalcik

The objective of Part II is to acquaint the reader with arithmetic operations using computer number systems. Most digital computers contain binary circuitry to perform arithmetic operations. In the TRS-80, the circuitry provides for addition and subtraction. Multiplication and division are performed by software routines in ROM. Internal computer operations are based on the principle of addition using the binary number system. Subtraction uses the two's complement method which is, in essence, an addition procedure. Multiplication uses repetitive addition, and division uses repetitive subtraction with the two's complement performed first for subtraction.

Computer programmers work with object programs and memory dumps, or printouts, to locate errors. An object program is a machine-language program, generated from a compiler or assembler, that can be executed by the computer directly. This often requires the performance of simple addition and subtraction of address values to find the data contents or computer instructions in computer memory. These addresses can be in octal or hexadecimal numbers, the shortcut notations of binary numbers. Due to the need for detailed error analysis, it may be necessary to add and subtract binary numbers.

Addition in General

The steps required to add two numbers are the same in all number systems. The general steps involved in the addition of numbers for any number system are shown in Table 1. This procedure is applicable to understanding addition of binary, octal, and hexadecimal values. All binary and octal digit symbols are also decimal symbols, so adding them as decimal digits is easy. Hexadecimal numbers however, which use some symbols quite different from decimal digits, are more complicated. Before any hexadecimal addition, or for that matter doing any arithmetic, you must convert the symbols A through F to their equivalent decimal values of 10 through 15.

Addition always starts with the rightmost column. When this column is greater than or equal to the value of the base, a "carry" occurs. Whenever a 1 is carried to the next column, the value being carried is equal to the value of the base. For example, if the sum of the numbers in the rightmost column is 17, a 1 is placed in the next column (to the left) which carries a value of 10

(the base). A 1 in the tens (power of 1) column is equal to 10 in the ones (power of 0) column. After the carry of 1, the remaining digit, in our case 7, is entered in the first column. Repeat these steps for each column.

Step 1 Add the first column (rightmost)
Step 2 If the column total from Step 1 is equal to or greater than the base, subtract the value of the base from the column total and carry 1 to the next column to the left.
Step 3 If there are additional columns, add the next column and repeat Step 2.

Table 1. *Steps in the addition process*

Adding Decimal Numbers

Let us add decimal 679 to decimal 964. Step 1 is to add the first column (rightmost). This addition of 9 to 4 gives a total of 13. Since 13 is greater than the highest single digit (9) in decimal, 1 is carried to the next column. Because the value of the base is 10, this value 10 is always carried, and a 3 is placed in the first column (13 – 10). Step 2 is to add the next column (tens column). Adding the digits 1, 7, and 6 gives a total of 14. Again a 1 is carried to the third column (hundreds column). Here, 14 minus 10 leaves a value of 4 for the tens column. The last column of 1, 6, and 9 totals 16. Again 16 minus 10 leaves 6 in the hundreds column, and a 1 is carried to the thousands column. The total number, therefore, is equal to 1,643. The comma for breaking up groups of three digits for readability is used only in the decimal system.

Adding Binary Numbers

The same addition steps used in adding decimal numbers can be applied to binary numbers. Refer to Figure 1 for specific addition and subtraction rules for binary numbers.

Binary Addition Rules

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 0 and carry 1, or it is 10_2 (two-bit number).

Binary Subtraction Rules

0 – 0 = 0
1 – 0 = 1
1 – 1 = 1
0 – 1 = 1 (with a borrow of 1)

Figure 1. *Four basic rules for addition and subtraction of binary numbers*

Let us add binary 110111 and binary 101110. The first step is to add the ones (power of 0) column of digits 1 and 0. The column total is 1, a single digit, so no carry occurs. The next step is to add the next column of 1 and 1, which gives 2. Since a decimal value of 2 cannot be used as a single digit, a carry occurs. To carry, place a 1 above the column to the left. The carry is 2, the value of the base. Since a value of 2 out of a column total of 2 is being carried, the remaining column value is 0. The third-column value is decimal 3. Again, a carry results. Since decimal 3 is greater than or equal to the base, we carry a 1 to the fourth column, leaving a 1 in the third column. The addition is the same for each column. The above solution of the binary addition example with the use of the expanded form for checking the results follows:

$$\begin{array}{r} 110111_2 \\ +\ 101110_2 \\ \hline 1100101_2 \end{array}$$

Expanded Form Check

$$\begin{aligned} (1\times32) + (1\times16) + (0\times8) + (1\times4) + (1\times2) + (1\times1) &= 55_{10} \\ (1\times32) + (0\times16) + (1\times8) + (1\times4) + (1\times2) + (0\times1) &= \underline{46_{10}} \\ & \quad 101_{10} \\ (1\times64) + (1\times32) + (0\times16) + (0\times8) + (1\times4) + (0\times2) + (1\times1) &= 101_{10} \end{aligned}$$

Adding Octal Numbers

To add the octal numbers 765 and 534, add the first column of 5 and 4, which totals 9. Since the maximum decimal value for a single digit in octal is 7, a carry must occur. A 1 is carried to the top of the next column, which carries a value equal to the base, 8. Because 8 of the column total of 9 has been carried, the difference of 1 is recorded below the first column. The second column of 1, 6, and 3 totals decimal 10. A carry of 1 to the third column carries a value of the base. Because 8 of the 10 in the second column has been carried, the difference of 2 is recorded below the second column. The last column of 1, 7, and 5 totals decimal 13. Since decimal 13 is greater than 8 a carry of 1 occurs, and the decimal 5 (13 minus 8) is recorded in the last column. The carry of 1 becomes the fourth octal digit. The following illustrates the completed octal addition problem with the use of the expanded form check:

Octal Addition

$$\begin{array}{r} 765_8 \\ +\ 534_8 \\ \hline 1521_8 \end{array}$$

Expanded Form Check

$$\begin{aligned} (7\times64) + (6\times8) + (5\times1) &= 501_{10} \\ (5\times64) + (3\times8) + (4\times1) &= \underline{348_{10}} \\ & \quad 849_{10} \\ (1\times512) + (5\times64) + (2\times8) + (1\times1) &= 849_{10} \end{aligned}$$

Adding Hexadecimal Numbers

This last addition example illustrates adding hexadecimal D86 to hexadecimal 6A9. For step 1, add 9 and 6 to obtain a total of decimal 15. Decimal 15 is represented as the hexadecimal symbol F and is recorded below the first column. The second column adds 8 and A. Change symbol A to decimal 10. The column total is equal to 18. Decimal 18 is greater than a single hexadecimal digit, therefore, a 1 is carried to the third column. This carries 16 (the base) of the total 18, and the remaining 2 is recorded below the second column. The third column of 1, 6, and D (decimal 13) equals a total of 20. Again, the column total exceeds the maximum size of the hexadecimal digit F (15), so a carry must occur. The 1 carry has the value of base 16, so a 4 is recorded for the third column. The last step is to simply record the 1 from the last carry as the fourth column. The following example shows the addition process:

Step 1	D86 + 6A9 F	F is decimal 15, thus no carry is required.
Step 2	^{1}D86 + 6A9 2F	In the second column, 8 is added to A (decimal 10) to yield 18. 18 minus 16 gives 2, with a carry of 1.
Step 3	^{1}D86$_{16}$ + 6A9$_{16}$ 142F$_{16}$	The third column of 1, D (decimal 13), and 6 gives 20, and 20 minus 16 equals 4 with a carry of 1 to the fourth column.

The solution of the above hexadecimal addition problem with the use of the expanded form check follows:

Hexadecimal Addition

$$\begin{array}{r} D86_{16} \\ 6A9_{16} \\ \hline 142F_{16} \end{array}$$

Expanded Form Check

$$\begin{array}{rrrrr} (13\times256) & + & (8\times16) & + & (6\times1) = 3{,}462_{10} \\ (6\times256) & + & (10\times16) & + & (9\times1) = 1{,}705_{10} \\ \hline & & & & 5{,}167_{10} \\ (1\times4096) & + & (4\times256) & + & (2\times16) + (15\times1) = 5{,}167_{10} \end{array}$$

If your results are not in agreement, check the hexadecimal addition. It is quite likely that the first attempts at addition are going to be failures. Common mistakes are forgetting to change the number symbols A, B, C, D, E, and F to their proper decimal equivalents, and failure to remember that carries involve the decimal number 16.

Subtraction in General

The principles of subtraction using decimal numbers can be applied to subtraction of numbers in any other base. Table 2 shows the two steps for subtraction which are repeated for each column of the numbers involved. Step 1, if the subtrahend digit of the column is larger than the minuend digit, is to borrow from the column on the left. The borrowed digit is always equal in decimal value to the base used. Thus, in binary subtraction, 2 is borrowed; in decimal, 10 is borrowed; in octal, 8 is borrowed; in hexadecimal, 16 is borrowed. Step 2 is to subtract the lower value from the top value. In the following paragraphs, the solutions of examples of subtractions using decimal, binary, octal, and hexadecimal numbers are given.

Subtracting Decimal Numbers

Let us demonstrate the general steps in subtracting decimal 485 from decimal 846. Step 1 is to determine if borrowing is necessary. Five is less than 6, so no borrowing is required. Taking 5 from 6 leaves 1, which is recorded in the first column. In the second column, subtracting 8 from 4 requires borrowing. A 1 is borrowed from the column to the left, leaving a value of 7 in the third column. Since the base is 10, 10 is the actual value being borrowed. The 4 in the second column is added to the borrowed 10, making 14. Taking 8 from 14 leaves 6. The third operation is to subtract 4 from the remaining 7 in the third column, giving 3. The following example shows the procedure for decimal subtraction:

Definitions: Subtrahend—number being subtracted
Minuend—number being subtracted from
Examples: In the operation 6 – 3, 3 is the subtrahend and 6 is the minuend.

Step 1 If the subtrahend digit of the column is larger than the minuend digit, borrow from the column to the left. When borrowing, remember to subtract 1 from the column being borrowed from. The value borrowed is always equal to the value of the base.

Step 2 Subtract the lower value from the top digit value. When the hexadecimal digit symbols of A through F are used, convert them to their equivalent decimal values.
Repeat Steps 1 and 2 for the next column to the left until there are no more columns.

Table 2. *Steps for subtraction*

Decimal Subtraction

846_{10} Minuend
$-\ 485_{10}$ Subtrahend
361_{10} Difference

The steps of subtraction described above can be applied easily to any

number system. In the next sections, we will apply the steps to do subtractions in the binary, octal, and hexadecimal systems.

Subtracting Binary Numbers

The problem is to subtract binary 01110 from binary 10101. Refer to Figure 1. The first column does not require borrowing, and the 0 is subtracted from the 1. In the second column we subtract 1 from 0, so borrowing must occur. A 1 is borrowed from the column to the left, leaving a 0 in the third column. The 1 borrowed from the third column becomes 2 in the second column because the base is 2. A 1 in the fours (power of 2) column is equal to 2 in the twos column. Continue the subtraction by taking 1 from 2 in the second column. In the third column, to subtract 1 from 0, borrowing is again required. The fourth column contains a 0, therefore nothing is available to borrow. Because of this, we borrow from the fifth column. This borrow of 1 from the fifth column changes the 0 in the fourth column to a 2. Recall that a 1 in the fifth (sixteens) column equals decimal value 2 in the fourth (eights) column. Now the fourth column has something available for borrowing. When the 1 in the column is borrowed, it leaves a 1 in the fourth column and becomes 2 in the third column. The subtraction of 1 from 2 yields a difference of 1. Subtraction of the fourth column, 1 minus 1, gives 0. Finally, the fifth column subtraction is 0 minus 0, giving 0. This example shows the subtraction discussed above and includes the expanded form check:

Binary Subtraction

$$\begin{array}{r} 10101_2 \\ -\ 01110_2 \\ \hline 00111_2 \end{array}$$

Expanded Form Check

$$\begin{array}{rl} (1\times16) + (0\times8) + (1\times4) + (0\times2) + (1\times1) = & 21_{10} \\ (0\times15) + (1\times8) + (1\times4) + (1\times2) + (0\times1) = & \underline{-14_{10}} \\ & 7_{10} \\ (0\times16) + (0\times8) + (1\times4) + (1\times2) + (1\times1) = & 7_{10} \end{array}$$

Subtracting Octal Numbers

If borrowing is necessary in subtracting octal numbers, the decimal equivalent of 8 is added to the column needing to borrow. Let us do the subtraction problem of taking 275 from 734. In the first column, we subtract 5 from 4, so borrowing is necessary. A 1 in the second column (eights column) is equal to 8 in the first (ones column). When 1 is borrowed from the second column, 1 is subtracted from the second column digit and 8 is added to the first column digit, 4, giving a total of decimal 12. Taking 5 from 12 results in a 7 for the first column value. In the second column, we subtract 7 from 2,

and again borrowing is necessary. Borrowing 1 from the third column (the sixty-fours column) adds 8 to the second column digit, 2, for a total of 10. Subtracting 7 from 10 results in a difference of 3. The last column has 2 to be taken away from 6, leaving 4. The following illustrates the octal subtraction problem solution with the use of the expanded form check:

Octal Problem

$$\begin{array}{r} 734_8 \\ -\ 275_8 \\ \hline 437_8 \end{array}$$

Expanded Form Check

$$\begin{array}{rcl} (7\times64) + (3\times8) + (4\times1) &=& 476_{10} \\ (2\times64) + (7\times8) + (5\times1) &=& -\ 189_{10} \\ & & \overline{\quad 287_{10}} \\ (4\times64) + (3\times8) + (7\times1) &=& 287_{10} \end{array}$$

Subtracting Hexadecimal Numbers

Subtracting in this number system is most easily performed by employing decimal equivalent values for the symbols of A through F as we did in the addition operation. Let's do the subtraction of hexadecimal 48F from A7B. The first column is the subtraction of F (decimal 15) from B (decimal 11). Borrowing is required. Borrowing 1 from the second column adds the base, 16, to the first column. Now we subtract 15 from 27, resulting in a value of 12 for the first column. This is converted to a hexadecimal C. In the second column, we subtract 8 from 6 (note, it is not 7 because we borrowed 1), and again we need to borrow. Borrowing 1 from the third column adds 16 to the second column which reduces A (value of 10) to a decimal 9. Then, 16 plus 6 gives a total of 22. Taking 8 from 22 leaves decimal 14, or hexadecimal E. Finally, in the last column, subtract 4 from 9 for a value of 5. The solution of this problem with the use of the expanded form check follows:

Hexadecimal Subtraction

$$\begin{array}{r} A7B_{16} \\ -\ 48F_{16} \\ \hline 5EC_{16} \end{array}$$

Expanded Form Check

$$\begin{array}{rcl} (10\times256) + (7\times16) + (11\times1) &=& 2683_{10} \\ (4\times256) + (8\times16) + (15\times1) &=& -\ 1167_{10} \\ & & \overline{\quad 1516_{10}} \\ (5\times256) + (14\times16) + (12\times1) &=& 1516_{10} \end{array}$$

Two's Complement

Subtraction can be carried out by the addition of complements, which means that only the hardware circuitry for the addition operation is used.

The meaning of the phrase "subtraction by addition of the complement" is made clear by the following example of decimal arithmetic. Suppose the six-digit decimal number 235,481 is to be subtracted from 684,673. This might be done conventionally:

$$684{,}673 - 235{,}481 = 449{,}192$$

The subtraction can also be done by the addition of the ten's complement as is shown in the next three computations.

$$684{,}673 + (1{,}000{,}000 - 235{,}481) - 1{,}000{,}000$$

The parenthesized term (1,000,000 – 235,481) is called the ten's (related to base 10) complement of 235,481. It yields:

$$\begin{array}{r} 1{,}000{,}000 \\ -\ 235{,}481 \\ \hline 764{,}519 \end{array} \quad \text{(ten's complement)}$$

Note that the ten's complement can be written down by inspection, by subtracting each digit of the number from 9, and then adding 1 to the low-order (least significant) digit at the right. Adding the ten's complement to the minuend we have:

$$\begin{array}{r} 684{,}673 \\ +\ 764{,}519 \\ \hline 1{,}449{,}192 \end{array}$$

To subtract 1,000,000 from the answer, it is necessary only to drop the high-order (most significant) digit at the left. The final sum is 449,192, which agrees with the result obtained earlier from conventional subtraction. The two's complement and one's complement in binary notation are analogous to the ten's complement and nine's complement of the decimal number. Take the two's complement of 00111001 by first subtracting as follows:

$$\begin{array}{r} 100000000 \\ -\ 00111001 \\ \hline 11000111 \end{array}$$

Note that this actual subtraction is not required, since the two's complement can be obtained by inspection of the number. Each bit of the binary number is simply inverted (1 is changed to 0, and 0 is changed to 1), and a 1 is added to the low-order (least significant) bit at the right. Circuits for inversion are simple.

The binary 00111001 was inverted providing the binary 11000110, then 1 was added to get 11000111, which is the two's complement of 00111001. In short, there are two steps in all binary cases.

1) Invert each digit of the binary number.
2) Add a 1 to the rightmost digit (least significant bit).

An example of the subtraction of a binary number is given below. The problem is to subtract decimal 3 from decimal 9 using a four-bit binary notation with two's complement.

9_{10}	Minuend	1001
-3_{10}	Subtrahend	−0011
6_{10}	Difference	?

Form the two's complement of the minuend by first changing the subtrahend 0011 to 1100 (by inverting) and then adding a 1 bit to the LSB. This yields 1101 as the two's complement of the subtrahend. Now perform addition as follows:

1001	(two's complement)
+ 1101	(two's complement)
10110	

The binary addition carries a bit into the fifth column which is ignored or dropped. The difference is 0110 or decimal 6. This checks with $9_{10} - 3_{10} = 6_{10}$ done previously.

Many microcomputers do have a distinct subtraction (hardware circuitry) operation. Then, of course, the basic laws of binary subtraction, as shown in Figure 1, are used instead of the two's complement procedure.

Conclusion

Computers, in general, operate in binary numbers, and to understand the working of a computer, you must understand the binary number system. A number expressed in binary numbers, or in its shortcut notations of octal and hexadecimal numbers, can also be expressed in decimal numbers. As long as the basic rules of arithmetic are observed, the result of any calculation leads to equivalent results.

UTILITY

TRSDOS Multiple Command Processor
Dandyzap
Slow Scroll

TRSDOS Multiple Command Processor

by Philip Sherman

A chained-command processor lets you enter a single command that then automatically executes a series of commands. NEWDOS/80 provides this option; TRSDOS doesn't. I soon tired of keying in the same set of DOS commands to start up my system; so I decided to find a way to set up a chained-command processor for TRSDOS.

My search started with Radio Shack's manuals. The Level II BASIC manual storage map shows that the keyboard has a DCB (device control block) located at 4012H (hexadecimal address). Locations 4016H and 4017H contain the address of the driver routine that is used whenever a character is to be read from the keyboard. When TRSDOS 2.3 is loaded from disk, the ROM driver address is changed to a TRSDOS driver that includes a keyboard debounce routine. The lowercase keyboard driver from Radio Shack also modifies the contents of 4016H and 4017H. I decided that by replacing the keyboard driver address with my own driver I could feed it a series of commands to be executed.

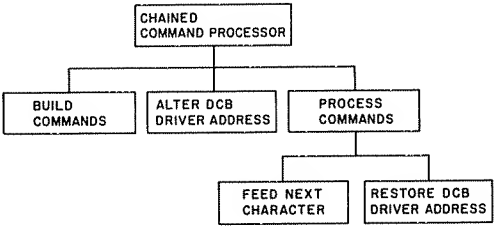


Figure 1. *Hierarchy of the chained-command processor*

Figure 1 is a hierarchy diagram of my chained-command processor. My requirement of a single command to start up my system forced me to write this program in assembly language. To simplify the initial version of the program, the BUILD COMMANDS function was replaced by a fixed list of commands. When you change the keyboard DCB driver address, you must save the old address so that it can be restored when the program is finished. The PROCESS COMMANDS function feeds the next character in the command string to whatever program requested the keyboard input. When the last character is fed, the original DCB driver address is restored, reenabling the keyboard.

Program Listing 1 is the assembler version of the multiple-command executer. Lines 130 through 170 save the existing driver address at 4016H and 4017H and replace it with the driver in lines 180 to 340. Figure 2 is a flowchart of the driver, which is also the PROCESS COMMANDS function of the Figure 1 hierarchy diagram.

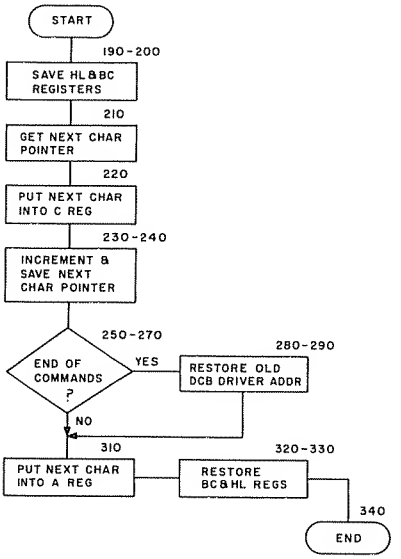


Figure 2. *Driver flowchart*

The ENTER key, when pressed, generates the code 13 (decimal). The code 24 is the end-of-commands indicator I chose. This code (shift and left arrow) is normally not used when entering commands because it erases the line being entered. The command list I used in the sample program starts BASIC, defines four disk file buffers, sets BASIC's memory size to the maximum available, and starts execution of a program called Menu. When I placed the program on my TRSDOS diskette, I gave it the filespec IPL/CMD. I set the AUTO (TRSDOS) function to invoke IPL when TRSDOS was initially loaded. Everything worked fine; I had a multiple-command processor that was invoked whenever TRSDOS was loaded. But, I detected two possible problem areas in my analysis of the technique. Since my command processor modified the keyboard driver address, I could not modify the driver address while the command processor was running. I also had to make sure that no program being loaded or executed would overlay the command processor program.

Changing the ORG statement (line 100) of Program Listing 1 will move the program to other memory locations. Use it to place the program where it won't be overlayed. Don't forget that BASIC has a work area at the top of

memory which it uses during its initialization. For a 32K or 48K RAM machine, setting the multiple-command processor load address 2K (800H) below the top of memory should be safe for most applications.

When I need to use a lowercase driver, I set the AUTO command to load the lowercase driver. When DOS's READY message appears, I enter the IPL command which then executes my list of commands.

The BASIC Program

Most software for the TRS-80 is written in BASIC. On my one-drive system, it is a chore to write software in assembly language and transfer it to a disk used for BASIC programs. I decided to rewrite the program in BASIC to make it easy to change the command list. Once again, I started with a hierarchy diagram of the program. Figure 3 shows the functional units of the BASIC version of the multiple-command processor. Program Listing 2 is the BASIC program written to implement the functions defined in Figure 3. I wrote the program in structured form to make debugging easier. All blocks of code are small enough that the entire block can be displayed on the monitor at once. This makes tracing detailed logic much easier.

Line 10 is a heading line giving the disk filespec name and the date of the last update. Lines 50 through 300 are the mainline processing routine.

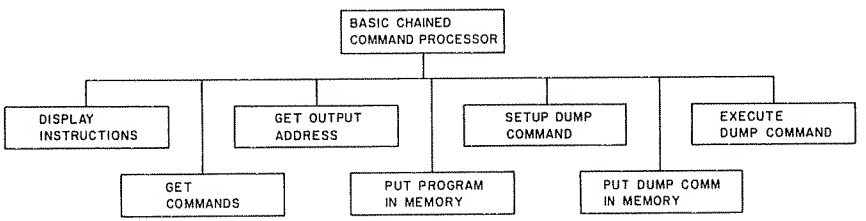


Figure 3. *Functions of the BASIC command processor*

Lines 50 and 60 set up string space, make BASIC treat all variables starting with the letter S as strings, and assign initial values to the variables to be used later.

Lines 70 through 90 display the title screen. Line 90 leaves the title screen on the video display while a counter is decremented 2000 times. The display is then cleared.

Lines 100 through 200 invoke the functions shown on the hierarchy diagram (see Figure 3). Each line has a REMark statement describing its function.

Line 280 moves the load address for the command processor down 256 bytes in memory and adds the ENTER and end-of-data characters to the DUMP command. Line 290 then causes the command processor to be POKEd into memory with the DUMP command as the command to be executed.

Line 300 sets the BASIC USR function to the starting address of the DUMP command version of the command processor. Once this is done, the USR function is invoked, transferring control to the command processor. The command processor will not return to the BASIC program.

Lines 3000 to 3100 get the string of commands to be executed. Since pressing ENTER tells the computer to read data, the INKEY$ function reads the keyboard directly. Figure 4 is the flowchart of this short but complex function. Remember that this function expects the first character to be available to it when it is invoked. Line 110 performs the read-a-character portion of the program once before starting this function.

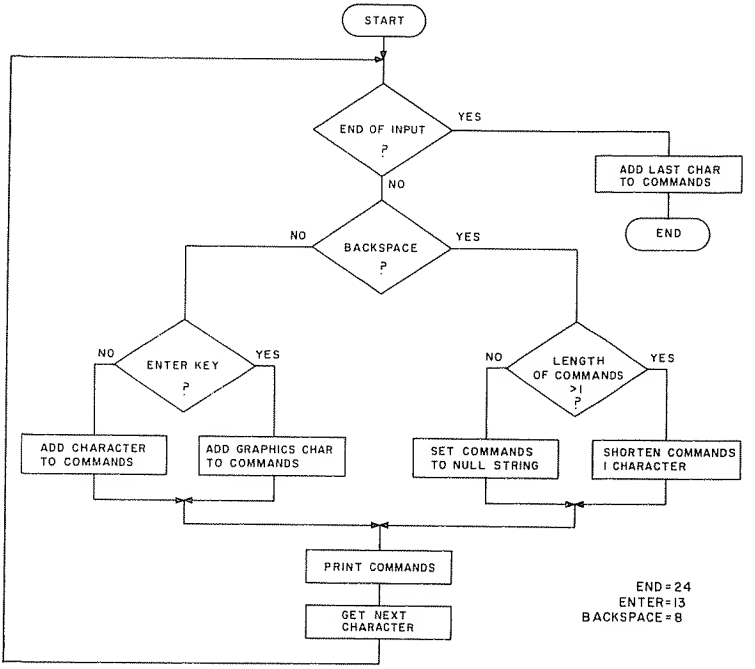


Figure 4. *Flowchart for lines 3000 to 3100 which read INKEY$ function directly from keyboard*

Line 3010 first checks for an end-of-data character. If it is not found, lines 3020 and 3030 process the character, then the next character is read from the keyboard. When the last character has been read, it is added to the end of the command string, and the input routine returns to its caller by executing the RETURN command.

Line 3020 will shorten the command string if a left arrow (code 8) key is read. If an ENTER key is read, it is replaced with a graphics block using the code 140. All other characters are appended to the end of the string being built.

Line 3030 erases the screen, writes the current command string, and returns to line 3010. Clearing the screen and rewriting the string is the simplest way to keep the current string displayed.

Lines 4000 to 4040 get the execution address of the command program. This address will be used to POKE the program and command string into memory.

Line 4010 initializes the string variable S to null. L is set to the default execution address of 47360. When L is initialized, the value chosen is the execution address divided by 256. The integer portion of this division is the high-order byte of the execution address.

Lines 4020 and 4030 display the instructions for entering the execution address. Line 4040 reads the desired execution address into string S. If no string is entered, the default execution address in L is used. If a string is entered, it is converted to numerics, divided by 256, and its integer portion replaces the default value in L.

Note: When a string is used for input, as in line 4040, you can enter non-numeric characters without BASIC catching them as errors. Since I designed this program to be used by an experienced person, I did not put in a code to validate the execution address supplied.

Lines 5000 through 5110 build the DUMP command used to place the program on disk. When this routine is invoked, variable L must be pointing to the starting (/256) address of the program to be dumped, and variable I must point to the last byte of the program being dumped. Line 5010 sets the SX string to the characters used in hexadecimal numbers.

Line 5020 sets the input variable for the hexadecimal converter, invokes the hex converter, and uses its output to build the hexadecimal starting address of the program to be dumped. Line 5030 uses the hex converter to get the low-order portion of the ending address to be dumped. The high-order portion of this address is set to the same value as the starting address to be dumped. The net effect of this is that the multiple-command program and the list of commands to be dumped must be no longer than 255 bytes. This is not a serious restriction, because it leaves 210 characters for the command list.

Either line 5040 or line 5045 builds the rest of the DUMP command. Line 5045 is not executed, because there is a quotation mark in the first byte of the line. If you are running this program under NEWDOS/80, remove the quotation mark to get the correct DUMP command.

Lines 5050 through 5070 tell you that the multiple-command program has been POKEd into memory and displays the DUMP command that has been built. If you don't want to continue the program execution, you can abort the program here by pressing the BREAK key.

Line 5080 requests the entry of an SX string. This is done to hold the messages on the display and permit using the BREAK key to stop the program.

Lines 5100 through 5110 contain the hexadecimal converter. The input is

passed to the converter in the variable M; the output is generated into the ST string variable. Line 5100 breaks the input number into two hex digits which will have values between 0 and 15.

Line 5110 uses these hexadecimal digits to index into the SX string and to get the hex character for the digit. Since BASIC does not let you get the character of a string, a zero adjusting factor of one is added to each hex digit. The hex digit values will now fall into the range of 1 to 16 which matches the length of the SX string.

Lines 6000 through 6090 POKE the control program and the commands to be executed into the selected memory locations. When this routine is invoked, L is used as a pointer to the starting location for POKEing, and SF must contain the list of commands to be POKEd. When it is finished, variable I will point to the last memory location being used by the multiple-command processor.

Line 6010 sets the initial value of I by multiplying L by 256. If I is greater than 32767, the POKE addresses must be expressed as negative numbers. The IF statement sets A to either zero or 65536 to make this adjustment. Line 6020 resets the data pointer to the first data statement in the program and reads the first byte to be POKEd into memory.

Line 6030 is the control structure for a DO-WHILE function. As long as J is less than 256, the routine in lines 6040 and 6050 will be invoked. J is initialized in line 6020 before the IF test is done in line 6030. When the value 256 is read into J, it indicates that the DO-WHILE function has been completed. The routine in lines 6040 and 6050 is bypassed with the GOTO 6060 statement.

Line 6040 handles relocation of addresses in the program being POKEd into memory. The value 190 occurs only as the high-order byte of an address. Since the program will always start on a 256-byte boundary, only this byte of addresses must be changed to relocate the program.

Line 6050 POKEs the program byte into memory. The constant A is used to change POKE addresses to negative numbers when the POKE addresses are greater than 32767. After a byte has been POKEd, the POKE address in I is incremented.

Lines 6060 through 6090 are a DO-UNTIL construct which POKEs the command list into memory. Line 6060, the FOR statement, uses the length of SF to determine how many characters are to be processed.

Line 6070 replaces the graphics block (140) with a 13 (ENTER) before POKEing it. Other characters are POKEd as they are. Line 6080 increments the storage address and closes the FOR statement. Line 6090 returns to the caller.

Lines 7000 through 7030 are DATA statements containing the machine-language code of the assembler program shown in Program Listing 1.

Lines 8000 through 8260 contain the instructions for running the pro-

gram. If you remove line 100, the instructions will not be displayed.

Notes

The TRSDOS DIR command does not stop reading characters from the multiple-command processor when the ENTER code is passed to it. The directions show that five spaces are needed after the ENTER key to process the next command properly. You may need to adjust the number of spaces if the next command is not processed properly. The multiple-command processor can be used to load a lowercase driver if it is the last command to be executed. Since the command processor restores the keyboard DCB driver address before executing the last command, the alteration of the driver address by the lowercase driver occurs after the command processor program has finished.

Program Listing 1. *Assembly-language multiple command processor*

```
BE00            00100           ORG     0BE00H
000D            00110 ENTER     EQU     13
0018            00115 DEND      EQU     24
4016            00120 DVRADR    EQU     4016H
BE00 2A1640     00130 BEGIN     LD      HL,(DVRADR)
BE03 2228BE     00140           LD      (OLDCB),HL      ;SAVE DRIVER ADDRE
     SS
BE06 210FBE     00150           LD      HL,DRVR
BE09 221640     00160           LD      (DVRADR),HL
BE0C C32D40     00170           JP      402DH
BE0F            00180 DRVR      EQU     $
BE0F E5         00190           PUSH    HL
BE10 C5         00200           PUSH    BC
BE11 2A2ABE     00210           LD      HL,(PTR)
BE14 4E         00220           LD      C,(HL)          ;GET NEXT CHAR
BE15 23         00230           INC     HL
BE16 222ABE     00240           LD      (PTR),HL        ;SAVE POINTER
BE19 7E         00250           LD      A,(HL)
BE1A FE18       00260           CP      DEND            ;END DATA?
BE1C 2006       00270           JR      NZ,MORE         ; NO
BE1E 2A28BE     00280           LD      HL,(OLDCB)
BE21 221640     00290           LD      (DVRADR),HL     ;RESTORE KB DRIVER

BE24            00300 MORE      EQU     $
BE24 79         00310           LD      A,C
BE25 C1         00320           POP     BC
BE26 E1         00330           POP     HL
BE27 C9         00340           RET
0002            00350 OLDCB     DEFS    2
BE2A 2CBE       00360 PTR       DEFW    CMDS
BE2C 56         00370 CMDS      DEFM    'VERIFY'
BE32 0D         00380           DEFB    ENTER
BE33 42         00390           DEFM    'BASIC'
BE38 0D         00400           DEFB    ENTER
BE39 34         00410           DEFM    '4'
BE3A 0D         00420           DEFB    ENTER
BE3B 0D         00430           DEFB    ENTER
BE3C 52         00440           DEFM    'RUN"MENU"'
BE45 0D         00450           DEFB    ENTER
BE46 18         00460           DEFB    DEND
BE00            00470           END     BEGIN
```

Program Listing 2. *BASIC multiple command processor*

```
10 REM IPL-03/25/81
50 CLEAR 300: DEFSTR S
60 I=0: J=0: K=0: S="": SF=S
70 CLS: PRINT "R/S TRSDOS MULTIPLE COMMAND GENERATOR"
80 PRINT: PRINT "WRITTEN MARCH, 1981 - PHILIP SHERMAN"
90 FOR I=1 TO 02000: NEXT: CLS
100 GOSUB8000:              REM DISPLAY INSTRUCTIONS
110 GOSUB 3100: GOSUB3010: REM GET INPUT STRING OF COMMANDS
120 GOSUB 4010:             REM GET EXECUTION ADDRESS
190 GOSUB 6010:             REM POKE PROGRAM AND CMDS INTO MEM
200 GOSUB 5010:             REM BUILD & DISPLAY DUMP COMMAND
280 L=L-1: SF=S+CHR$(140)+CHR$(24)
290 GOSUB 6010:             REM POKE DUMP COMMAND INTO MEMORY
300 DEFUSR=L*256-A: A=USR(1): REM EXECUTE DUMP COMMAND
310 '
3000 REM GET STRING OF COMMANDS
3010 IFASC(S)<>24THENGOSUB3020:GOSUB3100:GOTO3010ELSESF=SF+S:RETURN
3020 IFASC(S)=8THENIFLEN(SF)>1THENSF=LEFT$(SF,LEN(SF)-1)ELSESF=""ELSEI
     FASC(S)=13THENSF=SF+CHR$(140)ELSESF=SF+S
3030 CLS:PRINTSF;:RETURN
```

```
3100 S=INKEY$:IFS>""THENRETURNELSEGOTO3100
3110 '
4000 REM GET EXECUTION ADDRESS OF MULTIPLE COMMAND EXECUTER
4010 S="": L=185
4020 PRINT:PRINT"ENTER EXECUTION ADDRESS OF COMMAND FILE"
4030 PRINT"PRESS ENTER TO USE DEFAULT ADDRESS OF";L*256
4040 INPUT S: IF S="" THEN RETURN ELSE L=FIX(VAL(S)/256):RETURN
4050 '
5000 REM SETUP COMMAND TO WRITE PROGRAM TO DISK
5010 SX="0123456789ABCDEF"
5020 M=L: GOSUB5100   : SP=ST+"00"
5030 M=I-256*FIX(I/256): GOSUB 5100   : SE=LEFT$(SP,2)+ST
5040 S="DUMP IPL/CMD (START=X'"+SP+"',END=X'"+SE+"',TRA=X'"+SP+"')"
5045 'S="DUMP IPL/CMD:0 "+SP+"H "+SE+"H "+SP+"H"
5050 PRINT "COMMAND PROGRAM BUILT. PRESS ENTER TO PLACE IT ON DISK"
5060 PRINT "USING THE FOLLOWING DOS COMMAND"
5070 PRINT S
5080 INPUTSX: RETURN
5100 J=INT(M/16): M=M-16*J
5110 ST=MID$(SX,J+1,1)+MID$(SX,M+1,1): RETURN
5120 '
6000 REM POKE PROGRAM AND COMMAND(S) INTO MEMORY
6010 I=L*256: IF I>32767 THEN A=65536 ELSE A=0
6020 RESTORE: READ J
6030 IF J<256 THEN GOSUB 6040  : READ J: GOTO 6030  ELSE GOTO 6060
6040 IF J=190 THEN J=L
6050 POKE I-A,J: I=I+1: RETURN
6060 FOR J=1 TO LEN(SF)
6070 IF ASC(MID$(SF,J,1))=140 THEN POKE I-A,13 ELSE POKE I-A,ASC(MID$(
     SF,J,1))
6080 I=I+1: NEXT
6090 RETURN
6100 '
7000 DATA 42,22,64,34,40,190,33,15,190,34,22,64
7010 DATA 195,45,64
7020 DATA 229,197,42,42,190,78,35,34,42,190,126,254,24,32,6
7030 DATA 42,40,190,34,22,64,121,193,225,201,0,0,44,190,256
7040 '
8000 REM DISPLAY INSTRUCTIONS
8005 PRINT"This program will generate and write to disk a"
8010 PRINT"program for you that will execute multiple commands when"
8020 PRINT"you execute it. If you set the AUTO command to the output o
     f"
8030 PRINT"this BASIC program, the commands will be executed every tim
     e"
8040 PRINT"TRSDOS is loaded from disk."
8050 PRINT:PRINT
8060 PRINT"THE COMMANDS EXECUTED CANNOT INCLUDE ANY COMMAND THAT "
8070 PRINT"MODIFIES THE KEYBOARD DRIVER ADDRESS."
8080 PRINT: PRINT"This includes all known versions of lower case keybo
     ard drivers."
8090 PRINT:INPUT"Press enter key to continue";ST
8100 CLS:PRINT"When the screen clears, you can enter your input comman
     ds."
8110 PRINT"Enter them exactly as you would if you are starting your"
8120 PRINT"     system from the first DOSD READY."
8130 PRINT"Press the enter key exactly as you would do during startup.
     "
8140 PRINT"To end the input phase of the program, press the SHIFT and"

8150 PRINT"     backspace (";CHR$(93);") keys. ";CHR$(93);" will backs
     pace 1 character."
8160 PRINT"The execution address must be someplace in memory that will
     "
8170 PRINT"     not be overlayed by any program being executed."
8180 PRINT"The automatic write to disk routine requires the 256 bytes"

8190 PRINT"     of memory below the starting address you supply."
8200 PRINT"     This must not overlay any part of this BASIC program.
8210 PRINT"The default address is good for a 32k machine.
```

Program continued

```
8220 PRINT"Some sample commands are (";CHR$(140);" is the ENTER key):"

8230 ST=CHR$(140):PRINT"  VERIFY";ST;"DIR";ST;"     or  DIR";ST;"
     VERIFY";ST;"   (SPACES REQUIRED HERE)
8240 PRINT"  VERIFY";ST;"BASIC";ST;"4";ST;ST;"RUN";CHR$(34);"MENU";CHR
     $(34);ST
8250 INPUT"Press the enter key to start the program";SX
8260 CLS:RETURN
```

Dandyzap

by Richard T. Sornborger

The Dandyzap program is a TRSDOS-dependent utility with the capabilities of examining, modifying, or searching any desired bytes on a disk.

Both F3GUM and NV36 are master passwords for TRSDOS. With the master password in hand, you have access to any system file on any disk.

Getting Started

If you are using a disk-based editor assembler, type the code as shown in the Program Listing. If you have the tape version, you have to set ORG in line 190 greater than 6FFFH. You must then use the DUMP library command to get it onto disk. I wrote Dandyzap so you can relocate it by changing the ORG. No other modifications are necessary to relocate the program.

Operation

Mode 1 displays any given sector on a disk. When prompted, simply give the relative sector number of the desired sector. For example, to see Track 17, Sector 9, enter 179.

When you address the directory, you see a large graphics block at the base of the screen. This signifies that the sector is read-protected. Press ENTER to exit this function.

Mode 2 displays any given sector for a filespec. Simply reply with the desired filespec and give the relative sector within the file. Press ENTER to exit this function.

Mode 3 displays the encode for a password. Reply with the password you want. Type X to exit this function after the password has been generated. Pressing ENTER continues this mode.

Mode 4 displays the hashcode for a filespec. Simply respond with the desired filespec. Typing X aborts this function after the hashcode has been generated. Pressing ENTER continues this mode.

Mode 5 returns you to DOS READY. A reboot does not occur.

While you are in Modes 1 or 2, if a sector is being displayed, pressing the semicolon (;) key advances to you the next sector. The hyphen key (-) displays the preceding sector. The X key returns you to the main menu. The M key puts you in the modify mode. The S key puts you in the search mode. The T key toggles between drives 0 and 1. The T function is valuable when comparing two disks.

In the modify mode, responding with the desired relative byte within the sector after pressing the M key initiates a blinking cursor at the desired location. From this point, use the four arrow keys to guide the cursor. The relative position of the cursor is displayed in the lower left part of the screen. The Q key reads the sector again and aborts the modify mode; any valid key entry (0–9, A–F) is displayed in the sector. If the cursor is blinking on the second nibble of any given byte, only valid keys (0–9, A–F) are accepted. You must be on the first nibble of a byte to abort the modify mode or write to the disk.

If you are in the modify mode and you press the ENTER key, a mini-menu is displayed. The Y key writes the sector with the same read protection that was previously contained for that sector. The P key writes the sector read protected (as in a directory sector). The C key writes the sector non-read protected (as in any non-directory sector). The N key aborts the write function and displays the sector with modifications, if any, still intact.

Responding with the S key initiates the search mode. When you press the S, a large graphics block appears at the top left of the display. You may enter from one to four hex bytes that you want to search. If a match is found, a blinking cursor marks the match, and you immediately enter the modify mode. If you press ENTER after the S key, the search continues until the next occurrence is found or END OF FILE is encountered.

If you press an incorrect key, enter X to start over. If you wish to stop the search while it is in progress, press the X key. This ends the search and displays the sector at which the search was aborted.

This program fits into a 48K system with 8K to spare. It should fit into a 32K machine if you remove the comments.

Figure 1 shows two zaps to try on your TRSDOS disk. Remember to make a backup before you apply them.

```
Zap 1: To disable passwords
SYS2/SYS
FILE RELATIVE SECTOR 1
RELATIVE BYTE 5D
CHANGE: A2 61
TO: 96 42
Zap 2: To make BREAK more stable.
SYS1/SYS
FILE RELATIVE SECTOR 0
RELATIVE BYTE E4
CHANGE: 28
TO: 20
```

Figure 1. *Two TRSDOS zaps*

Program Listing. *Dandyzap*

Encyclopedia Loader

```
                00100 ;*****************************************************
                00110 ;*                DANDYZAP VERSION 1.0               *
                00120 ;*       WRITTEN BY,                                 *
                00130 ;*                  R.T SORNBORGER                   *
                00140 ;*                  268 TEMPLE WAY                   *
                00150 ;*                  VALLEJO, CALIF.                  *
                00160 ;*                        94590                      *
                00170 ;*       PHONE 707-553-1369                          *
                00180 ;*****************************************************
5200            00190           ORG     5200H
5200 21E958     00200 START     LD      HL,MENU         ;MENU OF COURSE...
5203 CD9C55     00210           CALL    DPLAY           ;DISPLAY IT !!
5206 CDDC5B     00220 MENSCN    CALL    KYBRD           ;READ KEYBOARD
5209 CD3300     00230           CALL    33H
520C 321956     00240           LD      (MSAVE),A       ;SAVE MODE
520F FE32       00250           CP      32H             ;WAS IT "2"
5211 CA4553     00260           JP      Z,DFS           ;IF SO GO TO FILE ROUTINE
5214 FE33       00270           CP      33H             ;WAS IT "3"
5216 CA0555     00280           JP      Z,DPH           ;GO TO PASSWORD HASHCODE
5219 FE34       00290           CP      34H             ;WAS IT "4"
521B CA4455     00300           JP      Z,DFHC          ;GO TO FILE HASHCODE
521E FE35       00310           CP      35H
5220 CA0044     00320           JP      Z,4400H         ;GO TO DOS
5223 FE31       00330           CP      31H
5225 20DF       00340           JR      NZ,MENSCN       ;IF INVALID THEN RE-SCAN
                00350 ;*****************************************************
                00360 ;*      DISPLAY DISK RELATIVE SECTOR ROUTINE         *
                00370 ;*****************************************************
5227 CDA655     00380 AB        CALL    MFCB            ;MOVE FILE CONTROL BLOCK
522A 21BE58     00390           LD      HL,DSN          ;DISK SECTOR QUESTION
522D CD9C55     00400           CALL    DPLAY           ;DISPLAY IT !!
5230 21F255     00410           LD      HL,INBUF        ;INPUT BUFFER LOCATION
5233 0603       00420 AB1       LD      B,03H           ;MAXIMUM 3 CHARACTERS
5235 CD4000     00430           CALL    40H             ;ROM INPUT ROUTINE
5238 7E         00440           LD      A,(HL)
5239 FE0D       00450           CP      0DH
523B 28C3       00460           JR      Z,START
523D CD5A1E     00470           CALL    1E5AH           ;CONVERT INPUT TO HEX
5240 ED53DC55   00480           LD      (AFCB+0AH),DE   ;SECTOR NUMBER TO FCB
5244 21D958     00490           LD      HL,DN           ;"DRIVE NUMBER QUESTION"
5247 CD9C55     00500           CALL    DPLAY           ;DISPLAY IT
524A 21F255     00510           LD      HL,INBUF
524D 0601       00520 AB2       LD      B,01H           ;1 CHARACTER INPUT
524F CD4000     00530           CALL    40H             ;ROM INPUT ROUTINE
5252 7E         00540           LD      A,(HL)          ;GET DRIVE NUMBER
5253 FE0D       00550           CP      0DH
5255 28F6       00560           JR      Z,AB2
5257 D630       00570           SUB     30H             ;SUB BIAS
5259 32D855     00580           LD      (AFCB+06H),A    ;DRIVE NUMBER TO FCB
525C 11D255     00590 REREAD    LD      DE,AFCB         ;POINT TO PSUEDO FCB
525F CD3644     00600           CALL    4436H           ;READ SECTOR
5262 321256     00610           LD      (PSAVE),A       ;SAVE READ STATUS
5265 CA9E53     00620           JP      Z,SK5
5268 FE06       00630           CP      06H             ;SECTOR READ PROTECTED ?
526A CA9E53     00640           JP      Z,SK5           ;JUMP IF ERRORCODE 6
526D 21435A     00650           LD      HL,CLS
5270 F5         00660           PUSH    AF
5271 CD9C55     00670           CALL    DPLAY
5274 F1         00680           POP     AF
5275 F6C0       00690           OR      0C0H            ;SET BITS 6 & 7
5277 CD0944     00700           CALL    4409H           ;DOS DISPLAY ERROR
527A 3A1956     00710           LD      A,(MSAVE)
527D FE32       00720           CP      32H
527F CA7F53     00730           JP      Z,SKIP5
5282 21CC57     00740           LD      HL,RETRY        ;"CARE TO RETRY MESSAGE"
5285 CD9C55     00750           CALL    DPLAY           ;DISPLAY MESSAGE
5288 21F255     00760 OVER1     LD      HL,INBUF        ;INPUT BUFFER
```

Program continued

```
528B 0601     00770          LD      B,1             ;ONE CHARACTER INPUT
528D CD4000   00780          CALL    40H             ;ROM'S INPUT ROUTINE
5290 7E       00790          LD      A,(HL)          ;GET INPUT KEY
5291 FE59     00800          CP      59H             ;TRY AGAIN (Y)
5293 2008     00810          JR      NZ,SK4          ;JUMP IF NOT "Y"
5295 213F5A   00820          LD      HL,CLMESS       ;CLEAR SCREEN OF ERROR
5298 CD9C55   00830          CALL    DPLAY           ;DISPLAY CLMESS
529B 18BF     00840          JR      REREAD          ;DO IT AGAIN...
529D FE4E     00850 SK4      CP      4EH             ;BACK TO MAIN MENU
529F CA0052   00860          JP      Z,START         ;JUMP IF "N"
52A2 18E4     00870          JR      OVER1           ;INVALID INPUT
              00880 ;*****************************************************
              00890 ;*              DISPLAY DRV, TRK, SEC                *
              00900 ;*****************************************************
52A4 E5       00910 SK1      PUSH    HL
52A5 D5       00920          PUSH    DE
52A6 C5       00930          PUSH    BC
52A7 CD1A56   00940          CALL    SCREEN          ;DISPLAY BUFFER TO SCREEN
52AA 11003C   00950          LD      DE,3C00H        ;DRV MESSAGE LOCATION
52AD 214A5A   00960          LD      HL,DRV          ;POINT TO MESSAGE
52B0 CDC357   00970          CALL    THREE           ;DISPLAY "DRV"
52B3 11403C   00980          LD      DE,3C40H        ;DRIVE NO. LOCATION
52B6 3AD855   00990          LD      A,(AFCB+06H)    ;GET ASCII DRIVE NO.
52B9 C630     01000          ADD     A,30H           ;MAKE IT ASCII
52BB 12       01010          LD      (DE),A          ;TO VIDEO...
52BC 11C03C   01020          LD      DE,3CC0H        ;TRK MESSAGE LOCATION
52BF 214D5A   01030          LD      HL,TRK          ;POINT TO MESSAGE
52C2 CDC357   01040          CALL    THREE           ;DISPLAY "TRK"
52C5 21003D   01050          LD      HL,3D00H        ;TRACK NO. LOCATION
52C8 3AED37   01060          LD      A,(37EDH)       ;GET TRACK NO.
52CB CDA056   01070          CALL    BTOA            ;CONVERT TO ASCII DECIMAL
52CE 71       01080          LD      (HL),C          ;MSB OF TRACK NO.
52CF 23       01090          INC     HL              ;BUMP VIDEO
52D0 70       01100          LD      (HL),B          ;LSB OF TRACK NO.
52D1 11803D   01110          LD      DE,3D80H        ;"SEC" MESSAGE LOCATION
52D4 21505A   01120          LD      HL,SEC          ;POINT TO MESSAGE
52D7 CDC357   01130          CALL    THREE           ;DISPLAY "SEC"
52DA 21C03D   01140          LD      HL,3DC0H        ;SECTOR NO. LOCATION
52DD 3AEE37   01150          LD      A,(37EEH)       ;GET SECTOR NO.
52E0 CDA056   01160          CALL    BTOA            ;CONVERT TO ASCII DECIMAL
52E3 71       01170          LD      (HL),C          ;MSB OF SECTOR NO.
52E4 23       01180          INC     HL              ;BUMP VIDEO
52E5 70       01190          LD      (HL),B          ;LSB OF SECTOR NO.
52E6 3A1956   01200          LD      A,(MSAVE)       ;WHICH MODE ARE WE IN ?
52E9 FE32     01210          CP      32H             ;DISPLAY FILE'S SECTOR ?
52EB 283E     01220          JR      Z,ABA           ;IF SO GO...
52ED CDF252   01230          CALL    ABD             ;DISPLAY "REL" MESS
52F0 1839     01240          JR      ABA
52F2 11403E   01250 ABD      LD      DE,3E40H
52F5 21565A   01260          LD      HL,REL          ;POINT TO "REL" MESSAGE
52F8 CDC357   01270          CALL    THREE           ;DISPLAY IT
52FB 21803E   01280 ABE      LD      HL,3E80H
52FE E5       01290          PUSH    HL
52FF 2ADC55   01300          LD      HL,(AFCB+0AH)   ;GET RELATIVE SECTOR
5302 3A1256   01310          LD      A,(PSAVE)       ;GET PROTECTION STATUS
5305 FE06     01320          CP      06H             ;IS IS THE DIRECTORY ?
5307 2801     01330          JR      Z,ABH           ;IF SO GO...
5309 2B       01340          DEC     HL              ;ADJUST RELATIVE SECTOR
530A AF       01350 ABH      XOR     A               ;CLEAR CARRY AND COUNTER
530B 010A00   01360          LD      BC,0AH
530E ED42     01370 ABB      SBC     HL,BC           ;SUB 10 TILL CARRY
5310 3803     01380          JR      C,ABC
5312 3C       01390          INC     A               ;BUMP "A" FOR EVERY SUB
5313 18F9     01400          JR      ABB             ;LOOP TILL CARRY
5315 09       01410 ABC      ADD     HL,BC           ;RESTORE "HL"
5316 EB       01420          EX      DE,HL           ;SAVE REMAINDER IN "DE"
5317 E1       01430          POP     HL              ;GET VIDEO LOCATION
5318 CDA056   01440          CALL    BTOA            ;CONVERT TO ASCII
531B 79       01450          LD      A,C
531C FE30     01460          CP      30H             ;IF "0" THEN BYPASS
```

```
531E 2802     01470          JR      Z,ABF
5320 71       01480          LD      (HL),C
5321 23       01490          INC     HL              ;BUMP VIDEO
5322 78       01500 ABF      LD      A,B
5323 70       01510          LD      (HL),B
5324 23       01520          INC     HL
5325 7B       01530 ABG      LD      A,E             ;GET REMAINDER
5326 CDA056   01540          CALL    BTOA            ;CONVERT TO ASCII
5329 70       01550          LD      (HL),B          ;TO VIDEO
532A C9       01560          RET
532B C1       01570 ABA      POP     BC
532C D1       01580          POP     DE
532D E1       01590          POP     HL
532E C9       01600          RET
532F CDA452   01610 SK2      CALL    SK1             ;DISPLAY SCREEN
5332 E5       01620          PUSH    HL
5333 D5       01630          PUSH    DE
5334 C5       01640          PUSH    BC
5335 11403E   01650          LD      DE,3E40H
5338 21535A   01660          LD      HL,FRS          ;POINT TO "FRS" MESSAGE
533B CDC357   01670          CALL    THREE           ;DISPLAY IT...
533E CDFB52   01680          CALL    ABE             ;DISPLAY FILE'S SECTOR
5341 C1       01690          POP     BC
5342 D1       01700          POP     DE
5343 E1       01710          POP     HL
5344 C9       01720          RET
              01730 ;*****************************************************
              01740 ;*              MODE 2 - DISPLAY FILES SECTOR        *
              01750 ;*****************************************************
5345 21F657   01760 DFS      LD      HL,FILESP       ;POINT TO "FILESPEC" MESS
5348 CD9C55   01770          CALL    DPLAY           ;DISPLAY IT
534B 0610     01780          LD      B,10H
534D 21D255   01790          LD      HL,AFCB
5350 CD4000   01800          CALL    40H             ;INPUT FILESPEC
5353 7E       01810          LD      A,(HL)
5354 FE0D     01820          CP      0DH
5356 CA0052   01830          JP      Z,START
5359 EB       01840          EX      DE,HL           ;INBUT BUFFER TO DE
535A 210060   01850          LD      HL,6000H        ;READ BUFFER LOCATION
535D 0600     01860          LD      B,00H           ;LRECL
535F CD2444   01870          CALL    4424H           ;OPEN FILE
5362 281B     01880          JR      Z,SKIP5         ;JUMP IF OPEN SUCCESSFUL
5364 F6C0     01890 ERRO     OR      0C0H
5366 CD0944   01900          CALL    4409H           ;DISPLAY ERROR
5369 21DC57   01910          LD      HL,CONT         ;CONTINUE MESSAGE
536C CD9C55   01920          CALL    DPLAY
536F CDDC5B   01930          CALL    KYBRD
5372 213F5A   01940          LD      HL,CLMESS
5375 CD9C55   01950          CALL    DPLAY
5378 3E1B     01960          LD      A,1BH
537A CD3300   01970          CALL    33H
537D 18C6     01980          JR      DFS
537F 211058   01990 SKIP5    LD      HL,RIF          ;REL SEC WITHIN FILE MESS
5382 CD9C55   02000          CALL    DPLAY
5385 0603     02010 DFS1     LD      B,3             ;3 CHARACTER INPUT MAX.
5387 21F255   02020          LD      HL,INBUF
538A CD4000   02030          CALL    40H
538D 7E       02040          LD      A,(HL)
538E FE0D     02050          CP      0DH
5390 28F3     02060          JR      Z,DFS1
5392 CD5A1E   02070          CALL    1E5AH
5395 21DC55   02080          LD      HL,AFCB+0AH
5398 73       02090          LD      (HL),E
5399 23       02100          INC     HL
539A 72       02110          LD      (HL),D
539B C35C52   02120          JP      REREAD
539E 3A1956   02130 SK5      LD      A,(MSAVE)       ;FIND MODE
53A1 FE32     02140          CP      32H             ;IS IT MODE 2 ?
53A3 2005     02150          JR      NZ,SK9
53A5 CD2F53   02160          CALL    SK2
```

Program continued

```
53A8 1803      02170          JR      SK10
53AA CDA452    02180 SK9      CALL    SK1
53AD 3A1256    02190 SK10     LD      A,(PSAVE)       ;GET PROTECTION-STATUS
53B0 FE06      02200          CP      06H             ;IS READ-PROTECTED ?
53B2 2008      02210          JR      NZ,SCAN         ;IF NOT THEN SCAN KEYBRD
53B4 21C03F    02220          LD      HL,3FC0H        ;FLAG DISPLAY LOCATION
53B7 368F      02230          LD      (HL),8FH        ;GRAPHIC FLAG
53B9 2C        02240          INC     L
53BA 368F      02250          LD      (HL),8FH
53BC CD2B00    02260 SCAN     CALL    2BH             ;SCAN KEYBOARD
53BF FE3B      02270          CP      3BH             ;IS IT ";"
53C1 282F      02280          JR      Z,INEXT         ;INCREMENT NEXT
53C3 FE2D      02290          CP      2DH             ;IS IT "-"
53C5 283D      02300          JR      Z,DNEXT         ;DECREMENT NEXT
53C7 FE58      02310          CP      58H             ;IS IT "X"
53C9 CA0052    02320          JP      Z,START         ;BACK TO MAIN MENU
53CC FE4D      02330          CP      4DH             ;IS IT "M"
53CE 284E      02340          JR      Z,MOD           ;GO TO MODIFY MODE
53D0 FE53      02350          CP      53H             ;IS IT "S"
53D2 CA595A    02360          JP      Z,SMODE         ;GO SEARCH
53D5 FE54      02370          CP      54H             ;IS IT "T"
53D7 2802      02380          JR      Z,TOGGLE        ;TOGGLE TO OPPOSITE DRIVE
53D9 18E1      02390          JR      SCAN            ;LOOP TILL KEY PRESSED
53DB 3AD855    02400 TOGGLE   LD      A,(AFCB+06H)    ;GET DRIVE NO.
53DE FE00      02410          CP      00H             ;IS IT DRIVE 0
53E0 2808      02420          JR      Z,TOG0
53E2 FE01      02430          CP      01H             ;IS IT DRIVE 1
53E4 20D6      02440          JR      NZ,SCAN         ;NOT 0 OR 1, THEN RESCAN
53E6 3E00      02450          LD      A,00H           ;MAKE IT DRIVE 0
53E8 1802      02460          JR      TOG1
53EA 3E01      02470 TOG0     LD      A,01H           ;MAKE IT DRIVE 1
53EC 32D855    02480 TOG1     LD      (AFCB+06H),A    ;PUT IN "FCB"
53EF C36357    02490          JP      QUIT            ;READ IN SECTOR
53F2 3A1256    02500 INEXT    LD      A,(PSAVE)       ;GET PROTECTION STATUS
53F5 FE06      02510          CP      06H             ;READ-PROTECTED ?
53F7 C25C52    02520          JP      NZ,REREAD       ;IF NOT THEN READ NEXT
53FA 2ADC55    02530          LD      HL,(AFCB+0AH)   ;GET NEXT
53FD 23        02540          INC     HL              ;BUMP NEXT
53FE 22DC55    02550          LD      (AFCB+0AH),HL   ;BACK TO FCB
5401 C35C52    02560          JP      REREAD          ;READ NEXT SECTOR
5404 2ADC55    02570 DNEXT    LD      HL,(AFCB+0AH)   ;GET NEXT
5407 7C        02580          LD      A,H             ;THIS PREVENTS
5408 B7        02590          OR      A               ;DECREMENTING
5409 2004      02600          JR      NZ,DNEXT2       ;PAST TRACK 0
540B 85        02610          ADD     A,L             ;       SECTOR 0...
540C 3D        02620          DEC     A
540D 2808      02630          JR      Z,DNEXT1
540F 3A1256    02640 DNEXT2   LD      A,(PSAVE)       ;GET PROTECTION STATUS
5412 FE06      02650          CP      06H             ;IS IT READ-PROTECTED ?
5414 2801      02660          JR      Z,DNEXT1        ;DEC NEXT ONE TIME IF SO
5416 2B        02670          DEC     HL              ;DECREMENT NEXT
5417 2B        02680 DNEXT1   DEC     HL              ;TWICE
5418 22DC55    02690          LD      (AFCB+0AH),HL   ;BACK TO FCB
541B C35C52    02700          JP      REREAD          ;READ PRECEEDING SECTOR
541E CDDC5B    02710 MOD      CALL    KYBRD           ;GET FIRST KEY
5421 CDAD56    02720          CALL    ATOHEX          ;CONVERT TO BINARY
5424 30F8      02730          JR      NC,MOD          ;IF INVALID...RETRY AGAIN
5426 4F        02740          LD      C,A             ;SAVE IN "C"
5427 CDDC5B    02750 MODO     CALL    KYBRD           ;GET SECOND KEY
542A CDAD56    02760          CALL    ATOHEX          ;CONVERT TO BINARY
542D 30F8      02770          JR      NC,MODO         ;INVALID KEY...TRY AGAIN
542F F5        02780          PUSH    AF              ;SAVE SECOND INPUT
5430 79        02790          LD      A,C             ;GET FIRST INPUT
5431 0F        02800          RRCA                    ;ALIGN TO HIGH 4 BITS
5432 0F        02810          RRCA
5433 0F        02820          RRCA
5434 0F        02830          RRCA
5435 4F        02840          LD      C,A             ;BACK TO "C"
5436 F1        02850          POP     AF              ;GET SECOND INPUT
5437 81        02860          ADD     A,C             ;2 KEYS BECOME 1
```

```
5438 5F       02870          LD      E,A             ;LSB OF BUFFER NOW
5439 1660     02880          LD      D,60H           ;MSB OF BUFFER
543B C3BB56   02890 MOD1     JP      BLINK           ;BLINK CURSOR
              02900 ;*****************************************************
              02910 ;*              WRITE A SECTOR ROUTINE               *
              02920 ;*****************************************************
543E E5       02930 WRITE    PUSH    HL
543F D5       02940          PUSH    DE
5440 21435A   02950          LD      HL,CLS
5443 CD9C55   02960          CALL    DPLAY
5446 212F58   02970          LD      HL,BESURE
5449 CD9C55   02980          CALL    DPLAY
544C CDDC5B   02990 WR3      CALL    KYBRD
544F FE59     03000          CP      59H
5451 2829     03010          JR      Z,WR4
5453 FE4E     03020          CP      4EH
5455 CA9E53   03030          JP      Z,SK5
5458 FE50     03040          CP      50H
545A 2813     03050          JR      Z,WR5
545C FE43     03060          CP      43H
545E 2806     03070          JR      Z,WR6
5460 FE52     03080          CP      52H
5462 2871     03090          JR      Z,RLO
5464 18E6     03100          JR      WR3
5466 3A1256   03110 WR6      LD      A,(PSAVE)
5469 FE06     03120          CP      06H
546B 200F     03130          JR      NZ,WR4
546D 1817     03140          JR      WR0
546F 3A1256   03150 WR5      LD      A,(PSAVE)
5472 FE06     03160          CP      06H
5474 C4C654   03170          CALL    NZ,DECFCB
5477 3E06     03180          LD      A,06H           ;READ-PROTECT FLAG
5479 321256   03190          LD      (PSAVE),A
547C 3A1256   03200 WR4      LD      A,(PSAVE)
547F FE06     03210          CP      06H
5481 284B     03220          JR      Z,FIX
5483 CDC654   03230          CALL    DECFCB
5486 3AD355   03240 WR0      LD      A,(AFCB+01H)
5489 CBF7     03250          SET     06H,A
548B 32D355   03260          LD      (AFCB+01H),A
548E 11D255   03270          LD      DE,AFCB
5491 CD3944   03280          CALL    4439H           ;WRITE A SECTOR
5494 F5       03290          PUSH    AF
5495 3EA8     03300          LD      A,0A8H          ;WRITE NON-READ PROT
5497 32E746   03310          LD      (46E7H),A
549A F1       03320          POP     AF
549B 2821     03330          JR      Z,WR1
549D F5       03340          PUSH    AF
549E 21435A   03350          LD      HL,CLS
54A1 CD9C55   03360          CALL    DPLAY
54A4 F1       03370          POP     AF
54A5 F6C0     03380          OR      0C0H
54A7 CD0944   03390          CALL    4409H
54AA 21CC57   03400          LD      HL,RETRY
54AD CD9C55   03410          CALL    DPLAY
54B0 CDDC5B   03420 WR2      CALL    KYBRD
54B3 FE4E     03430          CP      4EH
54B5 CABE54   03440          JP      Z,WR1
54B8 FE59     03450          CP      59H
54BA 28CA     03460          JR      Z,WR0
54BC 18F2     03470          JR      WR2
54BE CDC654   03480 WR1      CALL    DECFCB
54C1 D1       03490          POP     DE
54C2 E1       03500          POP     HL
54C3 C35C52   03510          JP      REREAD
54C6 2ADC55   03520 DECFCB   LD      HL,(AFCB+0AH)
54C9 2B       03530          DEC     HL
54CA 22DC55   03540          LD      (AFCB+0AH),HL
54CD C9       03550          RET
              03560 ;*****************************************************
```

Program continued

```
                03570 ;*              WRITE A SECTOR READ PROTECTED           *
                03580 ;*******************************************************
54CE 3EA9       03590 FIX     LD      A,0A9H          ;WRITE (READ-PROT) CMD
54D0 32E746     03600         LD      (46E7H),A       ;INSERT INTO DOS
54D3 18B1       03610         JR      WRO
                03620 ;*******************************************************
                03630 ;*              RELOCATE SECTOR ROUTINE                 *
                03640 ;*******************************************************
54D5 21BE58     03650 RLO     LD      HL,DSN
54D8 CD9C55     03660         CALL    DPLAY
54DB 21F255     03670         LD      HL,INBUF
54DE 0603       03680         LD      B,3
54E0 CD4000     03690         CALL    40H
54E3 7E         03700         LD      A,(HL)
54E4 FE0D       03710         CP      0DH
54E6 28ED       03720         JR      Z,RLO
54E8 CD5A1E     03730         CALL    1E5AH
54EB ED53DC55   03740         LD      (AFCB+0AH),DE
54EF 21D958     03750         LD      HL,DN
54F2 CD9C55     03760         CALL    DPLAY
54F5 21F255     03770         LD      HL,INBUF
54F8 0601       03780         LD      B,1
54FA CD4000     03790         CALL    40H
54FD 7E         03800         LD      A,(HL)
54FE D630       03810         SUB     30H
5500 32D855     03820         LD      (AFCB+06H),A
5503 1881       03830         JR      WRO
                03840 ;*******************************************************
                03850 ;*              DISPLAY PASSWORD HASHCODE               *
                03860 ;*******************************************************
5505 CD7155     03870 DPH     CALL    SYS2
5508 CD2750     03880         CALL    5027H           ;FILL NAME AREA W/ " "'S
550B 210358     03890         LD      HL,PSWORD
550E CD9C55     03900         CALL    DPLAY
5511 21F255     03910         LD      HL,INBUF
5514 0608       03920         LD      B,08H           ;NO MORE THAN 8 LONG
5516 CD4000     03930         CALL    40H
5519 CD6350     03940         CALL    5063H           ;CALL SYS2 HASH ROUTINE
551C 115551     03950         LD      DE,5155H
551F CDD150     03960         CALL    50D1H
5522 E5         03970         PUSH    HL              ;SAVE HASHCODE
5523 7D         03980         LD      A,L             ;GET LSB OF HASHCODE
5524 CD8256     03990         CALL    HEXCV
5527 7C         04000         LD      A,H             ;GET ASCII MSB OF HASH
5528 CD3300     04010         CALL    33H             ;DISPLAY MSB
552B 7D         04020         LD      A,L             ;GET ASCII LSB OF HASH
552C CD3300     04030         CALL    33H
552F E1         04040         POP     HL
5530 7C         04050         LD      A,H             ;GET MSB OF HASHCODE
5531 CD8256     04060         CALL    HEXCV
5534 7C         04070         LD      A,H             ;GET MSB OF HASH
5535 CD3300     04080         CALL    33H
5538 7D         04090         LD      A,L
5539 CD3300     04100         CALL    33H
553C 3E0A       04110         LD      A,0AH
553E CD3300     04120         CALL    33H
5541 C37555     04130         JP      SCFX
                04140 ;*******************************************************
                04150 ;*              DISPLAY FILES HASHCODE                  *
                04160 ;*******************************************************
5544 CD7155     04170 DFHC    CALL    SYS2
5547 21F657     04180         LD      HL,FILESP
554A CD9C55     04190         CALL    DPLAY
554D 060C       04200         LD      B,12
554F 21F255     04210         LD      HL,INBUF
5552 CD4000     04220         CALL    40H
5555 CD2750     04230         CALL    5027H           ;CALL SYS2/SYS
5558 215D51     04240         LD      HL,515DH        ;POINT TO FILESPEC
555B CD9B50     04250         CALL    509BH
555E CD8256     04260         CALL    HEXCV
```

```
5561 7C       04270          LD      A,H
5562 CD3300   04280          CALL    33H
5565 7D       04290          LD      A,L
5566 CD3300   04300          CALL    33H
5569 3E0A     04310          LD      A,0AH
556B CD3300   04320          CALL    33H
556E C37555   04330          JP      SCFX
5571 3ED4     04340 SYS2     LD      A,0D4H          ;LOADER CODE
5573 EF       04350          RST     28H             ;LOAD SYS2/SYS
5574 C9       04360          RET                     ;BACK TO CALLING ROUTINE
5575 21DC57   04370 SCFX     LD      HL,CONT         ;CONTINUE MESSAGE
5578 CD9C55   04380          CALL    DPLAY
557B CDDC5B   04390 SCFX1    CALL    KYBRD           ;GET A KEY
557E FE58     04400          CP      58H             ;IS IT "X"
5580 CA0052   04410          JP      Z,START         ;BACK TO MENU
5583 FE0D     04420          CP      0DH             ;IS IT "ENTER"
5585 20F4     04430          JR      NZ,SCFX1        ;GET A KEY IF NOT
5587 213F5A   04440          LD      HL,CLMESS
558A CD9C55   04450          CALL    DPLAY
558D 3E1B     04460          LD      A,1BH
558F CD3300   04470          CALL    33H
5592 3A1956   04480          LD      A,(MSAVE)       ;WHICH MODE ???
5595 FE33     04490          CP      33H             ;IS IT FILE HASH
5597 CA0555   04500          JP      Z,DPH           ;IF SO GO...
559A 18A8     04510          JR      DFHC            ;GO TO PSWRD HASH
              04520 ;*****************************************************
              04530 ;***              DISPLAY ROUTINE                  ***
              04540 ;*****************************************************
559C 7E       04550 DPLAY    LD      A,(HL)          ;GET CHAR. TO DISPLAY
559D FE03     04560          CP      03H             ;IS IT THE DELIMITER ?
559F C8       04570          RET     Z               ;RETURN IF SO...
55A0 CD3A03   04580          CALL    33AH            ;CALL ROM DISPLAY ROUTINE
55A3 23       04590          INC     HL              ;BUMP MESSAGE POINTER
55A4 18F6     04600          JR      DPLAY           ;LOOP TILL DELIMITER
              04610 ;*****************************************************
              04620 ;***    MOVE "FCB" ROUTINE AND SAVE ORIGINAL "FCB"    ***
              04630 ;*****************************************************
55A6 21B255   04640 MFCB     LD      HL,FCB          ;POINT TO "FCB"
55A9 11D255   04650          LD      DE,AFCB         ;ACTUAL LOCATION OF "FCB"
55AC 012000   04660          LD      BC,32           ;ALL 32 BYTES OF IT...
55AF EDB0     04670          LDIR                    ;MOVE IT !!!
55B1 C9       04680          RET                     ;RETURN TO CALLER...
              04690 ;*****************************************************
              04700 ;***                " F  C  B "                    ***
              04710 ;*****************************************************
55B2 80       04720 FCB      DEFB    80H
55B3 20       04730          DEFB    20H
55B4 00       04740          DEFB    00H
55B5 00       04750          DEFB    00H
55B6 60       04760          DEFB    60H
55B7 00       04770          DEFB    00H
55B8 00       04780          DEFB    00H
55B9 00       04790          DEFB    00H
55BA 00       04800          DEFB    00H
55BB 00       04810          DEFB    00H
55BC 00       04820          DEFB    00H
55BD 00       04830          DEFB    00H
55BE FF       04840          DEFB    0FFH
55BF FF       04850          DEFB    0FFH
55C0 00       04860          DEFB    00H
55C1 1F       04870          DEFB    1FH
55C2 20       04880          DEFB    20H
55C3 00       04890          DEFB    00H
55C4 10       04900          DEFB    10H
55C5 1F       04910          DEFB    1FH
55C6 40       04920          DEFB    40H
55C7 00       04930          DEFB    00H
55C8 20       04940          DEFB    20H
55C9 1F       04950          DEFB    1FH
55CA 60       04960          DEFB    60H
```

Program continued

```
55CB 00       04970          DEFB    00H
55CC 30       04980          DEFB    30H
55CD 1F       04990          DEFB    1FH
55CE 80       05000          DEFB    80H
55CF 00       05010          DEFB    00H
5500 40       05020          DEFB    40H
5501 1F       05030          DEFB    1FH
0020          05040 AFCB     DEFS    32              ;RESERVED FOR ACTUAL FCB
0020          05050 INBUF    DEFS    32              ;INPUT BUFFER LOCATION
5612 00       05060 PSAVE    DEFB    00H             ;READ PROTECT STATUS
5613 00       05070 DSAVE    DEFB    00H             ;ASCII DRIVE NUMBER
5614 00       05080 ESAVE    DEFB    00H             ;RB OF SECTOR BUFFER
5615 0000     05090 NSAVE    DEFW    00H
5617 0000     05100 RBSAVE   DEFW    00H
5619 00       05110 MSAVE    DEFB    00H
              05120 ;*****************************************************
              05130 ;*            SCREEN DISPLAY ROUTINE                 *
              05140 ;*****************************************************
561A CDC901   05150 SCREEN   CALL    01C9H           ;CLS
561D 21073C   05160          LD      HL,3C07H        ;DISPLAY LOCATION
5620 110060   05170          LD      DE,6000H        ;BUFFER LOCATION
5623 011010   05180          LD      BC,1010H
5626 D5       05190 SC0      PUSH    DE
5627 C5       05200          PUSH    BC
5628 1A       05210 S1       LD      A,(DE)          ;READ A BYTE
5629 C5       05220          PUSH    BC              ;SAVE COUNTER
562A D5       05230          PUSH    DE              ;SAVE BUFFER POINTER
562B E5       05240          PUSH    HL              ;SAVE VIDEO POINTER
562C CD8256   05250          CALL    HEXCV           ;CONVERT BUFFER TO ASCII
562F 54       05260          LD      D,H             ;HIGH ASCII BYTE TO "D"
5630 5D       05270          LD      E,L             ;LOW ASCII BYTE TO "E"
5631 E1       05280          POP     HL              ;RESTORE VIDEO POINTER
5632 72       05290          LD      (HL),D
5633 23       05300          INC     HL
5634 73       05310          LD      (HL),E
5635 D1       05320          POP     DE              ;RESTORE BUFFER POINTER
5636 C1       05330          POP     BC              ;RESTORE COUNTER
5637 23       05340          INC     HL              ;BUMP VIDEO
5638 13       05350          INC     DE              ;BUMP BUFFER
5639 78       05360          LD      A,B             ;GET COUNTER
563A CB2F     05370          SRA     A               ;BUMP VIDEO 2 TIMES
563C 3001     05380          JR      NC,S0           ;FOR EVERY 4 DECS OF "B"
563E 23       05390          INC     HL
563F 10E7     05400 S0       DJNZ    S1
5641 C1       05410          POP     BC              ;RESTORE COUNTER
5642 D1       05420          POP     DE              ;RESTORE BUFFER POINTER
5643 1A       05430 SC1      LD      A,(DE)          ;READ A BYTE
5644 FE21     05440          CP      21H             ;FIND BYTE RANGE
5646 3804     05450          JR      C,DOT           ;JUMP IF NON-DISPLAYABLE
5648 FEC0     05460          CP      0C0H            ;CHECK HIGH-RANGE
564A 3802     05470          JR      C,S2            ;JUMP IF WITHIN RANGE
564C 3E2E     05480 DOT      LD      A,2EH           ;CHANGE TO "."
564E 77       05490 S2       LD      (HL),A          ;BYTE TO SCREEN
564F 23       05500          INC     HL              ;BUMP VIDEO
5650 1C       05510          INC     E               ;BUMP BUFFER
5651 280F     05520          JR      Z,D0            ;JUMP IF BUFFER EMPTY
5653 10EE     05530          DJNZ    SC1             ;LOOP THRU LINE
5655 7D       05540          LD      A,L             ;GET LSB OF VIDEO
5656 E6C0     05550          AND     0C0H
5658 C647     05560          ADD     A,47H           ;BUMP TO NEXT LINE
565A 6F       05570          LD      L,A
565B 3001     05580          JR      NC,S5
565D 24       05590          INC     H
565E 0610     05600 S5       LD      B,10H
5660 18C4     05610          JR      SC0             ;LOOP TILL DONE
5662 11043C   05620 D0       LD      DE,3C04H        ;NUMBERED COLUMN START
5665 3E00     05630          LD      A,00H           ;FIRST NUMBER
5667 F5       05640 SK6      PUSH    AF              ;SAVE NUMBER
5668 CD8256   05650          CALL    HEXCV           ;CONVERT TO ASCII
566B EB       05660          EX      DE,HL
```

```
566C 72       05670          LD      (HL),D          ;MSB OF ASCII BYTE
566D 23       05680          INC     HL              ;BUMP VIDEO
566E 73       05690          LD      (HL),E          ;LSB OF ASCII BYTE
566F 23       05700          INC     HL
5670 36AA     05710          LD      (HL),0AAH       ;VERT. LINE DOWN SCREEN
5672 EB       05720          EX      DE,HL
5673 7B       05730          LD      A,E             ;LSB OF VIDEO
5674 E6C0     05740          AND     0C0H            ;BACK TO START OF LINE
5676 C644     05750          ADD     A,44H           ;BUMP ONE LINE DOWN
5678 3001     05760          JR      NC,SK7          ;IF NO OVERFLOW JUMP
567A 14       05770          INC     D               ;BUMP MSB OF VIDEO
567B 5F       05780 SK7      LD      E,A             ;BACK TO DE
567C F1       05790          POP     AF              ;RESTORE NUMBER
567D C610     05800          ADD     A,10H           ;NEXT NUMBER
567F D8       05810          RET     C               ;RETURN IF OVERFLOW...
5680 18E5     05820          JR      SK6
              05830 ;*****************************************************
              05840 ;*            BINARY TO ASCII CONV ROUTINE           *
              05850 ;*****************************************************
5682 4F       05860 HEXCV    LD      C,A             ;SAVE 2 HEX DIGITS
5683 CB3F     05870          SRL     A               ;ALIGN HIGH DIGIT
5685 CB3F     05880          SRL     A
5687 CB3F     05890          SRL     A
5689 CB3F     05900          SRL     A
568B CD9756   05910          CALL    TEST            ;CONVERT TO ASCII
568E 67       05920          LD      H,A             ;SAVE HIGH
568F 79       05930          LD      A,C             ;RESTORE ORIGINAL
5690 E60F     05940          AND     0FH             ;ALIGN LOW DIGIT
5692 CD9756   05950          CALL    TEST            ;CONVERT TO ASCII
5695 6F       05960          LD      L,A             ;SAVE LOW
5696 C9       05970          RET
5697 C630     05980 TEST     ADD     A,30H           ;ADD ASCII BIAS
5699 FE3A     05990          CP      3AH             ;TEST FOR 0-9
569B 3802     06000          JR      C,TEST1         ;OF IF 0-9
569D C607     06010          ADD     A,7             ;ADJUST FOR A-F
569F C9       06020 TEST1    RET
              06030 ;*****************************************************
              06040 ;*      CONVERT BINARY TO 2 DECIMAL ASCII DIGITS      *
              06050 ;*****************************************************
56A0 0E30     06060 BTOA     LD      C,30H           ;ASCII "0"
56A2 D60A     06070 BTOA1    SUB     0AH             ;SUB 10 TILL CARRY
56A4 3803     06080          JR      C,BTOA2         ;IF CARRY THEN JUMP
56A6 0C       06090          INC     C               ;BUMP LOW ASCII DIGIT
56A7 18F9     06100          JR      BTOA1           ;LOOP TILL CARRY
56A9 C63A     06110 BTOA2    ADD     A,3AH           ;ADJUST MSB
56AB 47       06120          LD      B,A             ;HIGH ASCII DIGIT TO "B"
56AC C9       06130          RET                     ;BC=2 ASCII NO.S
              06140 ;*****************************************************
              06150 ;*              ASCII TO HEX INPUT                   *
              06160 ;*****************************************************
56AD D630     06170 ATOHEX   SUB     30H             ;REMOVE ASCII BIAS
56AF FE0A     06180          CP      0AH             ; 0-9 ?
56B1 D8       06190          RET     C               ;RETURN IF 0-9
56B2 D611     06200          SUB     11H             ; A-F?
56B4 FE06     06210          CP      06H             ;GREATER THAN F?
56B6 D0       06220          RET     NC
56B7 C60A     06230          ADD     A,0AH
56B9 37       06240          SCF
56BA C9       06250          RET
              06260 ;*****************************************************
              06270 ;*      BLINKING CURSOR AND KEYBOARD SCAN ROUTINE      *
              06280 ;*****************************************************
56BB 7B       06290 BLINK    LD      A,E
56BC CD8256   06300          CALL    HEXCV
56BF D5       06310          PUSH    DE
56C0 EB       06320          EX      DE,HL
56C1 21003F   06330          LD      HL,3F00H
56C4 72       06340          LD      (HL),D
56C5 23       06350          INC     HL
56C6 73       06360          LD      (HL),E
```

Program continued

```
56C7 D1        06370          POP     DE
56C8 21073C    06380          LD      HL,3C07H
56CB 7B        06390          LD      A,E
56CC CB3F      06400          SRL     A
56CE CB3F      06410          SRL     A
56D0 CB3F      06420          SRL     A
56D2 CB3F      06430          SRL     A
56D4 47        06440          LD      B,A
56D5 04        06450          INC     B
56D6 7D        06460          LD      A,L
56D7 1003      06470 BL2      DJNZ    BL3
56D9 6F        06480          LD      L,A
56DA 1807      06490          JR      BL4
56DC C640      06500 BL3      ADD     A,40H
56DE 30F7      06510          JR      NC,BL2
56E0 24        06520          INC     H
56E1 18F4      06530          JR      BL2
56E3 7B        06540 BL4      LD      A,E
56E4 E60F      06550          AND     0FH             ;SAVE BITS 0-3
56E6 280B      06560          JR      Z,SRA1          ;IF LOW BYTE=0 JUMP
56E8 47        06570          LD      B,A             ;TO "B" FOR DJNZ
56E9 78        06580 BL5      LD      A,B             ;GET COUNTER
56EA CB2F      06590          SRA     A               ;INC L IF NO CARRY.
56EC 3801      06600          JR      C,SRA0
56EE 2C        06610          INC     L               ;ADJUST FOR SPACE
56EF 2C        06620 SRA0     INC     L
56F0 2C        06630          INC     L
56F1 10F6      06640          DJNZ    BL5             ;IF B=0 THEN HL= CORRECT
               06650                                  ;LINE AND COLUMN OF
               06660                                  ;SELECTED BYTE...
56F3 CDF856    06670 SRA1     CALL    BL6
56F6 1828      06680          JR      BL8
56F8 D5        06690 BL6      PUSH    DE              ;SAVE BUFFER POINTER
56F9 E5        06700          PUSH    HL              ;SAVE CURSOR POSITION
56FA 4E        06710          LD      C,(HL)          ;SAVE DISPLAY BYTE
56FB 0600      06720 TWOBEE   LD      B,0             ;256 LOOPS
56FD CD2B00    06730          CALL    2BH             ;SCAN KEYBOARD
5700 B7        06740          OR      A               ;SET FLAG IF KEY PRESSED
5701 200D      06750          JR      NZ,BL7          ;JUMP IF KEY PRESSED
5703 10F8      06760          DJNZ    TWOBEE+2
5705 CB7E      06770          BIT     7,(HL)          ;IS CURSOR PRESENT ?
5707 2004      06780          JR      NZ,CURSOR       ;JUMP IF NOT
5709 368F      06790          LD      (HL),8FH        ;DISPLAY CURSOR
570B 18EE      06800          JR      TWOBEE
570D 71        06810 CURSOR   LD      (HL),C          ;DISPLAY BYTE
570E 18EB      06820          JR      TWOBEE
5710 E1        06830 BL7      POP     HL
5711 D1        06840          POP     DE
5712 368F      06850          LD      (HL),8FH        ;DISPLAY CURSOR
5714 C5        06860          PUSH    BC              ;SAVE BYTE AT CURSOR POS.
5715 F5        06870          PUSH    AF              ;SAVE INPUT KEY
5716 010010    06880          LD      BC,1000H
5719 CD6000    06890          CALL    60H             ;WASTE TIME
571C F1        06900          POP     AF
571D C1        06910          POP     BC
571E 71        06920          LD      (HL),C          ;ORIG. BYTE BACK TO VIDEO
571F C9        06930          RET
5720 FE09      06940 BL8      CP      09H             ;IS IT RIGHT ARROW
5722 2829      06950          JR      Z,RIGHT
5724 FE08      06960          CP      08H             ;IS IT LEFT ARROW ?
5726 2829      06970          JR      Z,LEFT
5728 FE5B      06980          CP      5BH             ;IS IT UP ARROW ?
572A 2829      06990          JR      Z,UP
572C FE0A      07000          CP      0AH             ;IS IT DOWN ARROW ?
572E 282C      07010          JR      Z,DOWN
5730 FE51      07020          CP      51H             ;IS IT "Q"
5732 282F      07030          JR      Z,QUIT
5734 FE58      07040          CP      58H             ;IS IT "X"
5736 283C      07050          JR      Z,EX
5738 FE53      07060          CP      53H             ;IS IT "S"
```

```
573A CA595A   07070          JP      Z,SMODE         ;GO TO SEARCH ROUTINE
573D FE0D     07080          CP      0DH
573F CA3E54   07090          JP      Z,WRITE
5742 47       07100          LD      B,A             ;SAVE INPUT KEY
5743 CDAD56   07110          CALL    ATOHEX
5746 382F     07120          JR      C,VALID
5748 CDF856   07130          CALL    BL6
574B 18D3     07140          JR      BL8
574D 1C       07150 RIGHT    INC     E               ;BUMP BUFFER NUMBER
574E C3BB56   07160          JP      BLINK
5751 1D       07170 LEFT     DEC     E               ;DEC BUFFER NUMBER
5752 C3BB56   07180          JP      BLINK
5755 7B       07190 UP       LD      A,E             ;GET BUFFER NUMBER
5756 D610     07200          SUB     10H             ;SUBTRACT 16
5758 5F       07210          LD      E,A
5759 C3BB56   07220          JP      BLINK
575C 7B       07230 DOWN     LD      A,E             ;GET BUFFER NUMBER
575D C610     07240          ADD     A,10H           ;ADD 16
575F 5F       07250          LD      E,A
5760 C3BB56   07260          JP      BLINK
5763 3A1256   07270 QUIT     LD      A,(PSAVE)       ;DIRECTORY SECTOR ?
5766 FE06     07280          CP      06H
5768 2807     07290          JR      Z,Q1            ;IF SO JUMP
576A 2ADC55   07300          LD      HL,(AFCB+0AH)   ;GET NEXT
576D 2B       07310          DEC     HL              ;BACK TO ORIGINAL SECTOR
576E 22DC55   07320          LD      (AFCB+0AH),HL   ;BACK TO "FCB"
5771 C35C52   07330 Q1       JP      REREAD          ;REREAD ORIGINAL SECTOR
5774 C30052   07340 EX       JP      START
5777 F5       07350 VALID    PUSH    AF              ;GET INPUT KEY
5778 78       07360          LD      A,B
5779 77       07370          LD      (HL),A
577A 23       07380          INC     HL
577B D5       07390          PUSH    DE
577C CDF856   07400 SECOND   CALL    BL6
577F 47       07410          LD      B,A
5780 CDAD56   07420          CALL    ATOHEX
5783 4F       07430          LD      C,A
5784 30F6     07440          JR      NC,SECOND
5786 D1       07450          POP     DE
5787 78       07460          LD      A,B
5788 77       07470          LD      (HL),A
5789 F1       07480          POP     AF
578A 87       07490          ADD     A,A
578B 87       07500          ADD     A,A
578C 87       07510          ADD     A,A
578D 87       07520          ADD     A,A
578E 81       07530          ADD     A,C
578F 12       07540          LD      (DE),A
5790 E5       07550          PUSH    HL
5791 D5       07560          PUSH    DE
5792 C5       07570          PUSH    BC
5793 21003C   07580          LD      HL,3C00H
5796 7B       07590          LD      A,E
5797 CB3F     07600          SRL     A
5799 CB3F     07610          SRL     A
579B CB3F     07620          SRL     A
579D CB3F     07630          SRL     A
579F B7       07640          OR      A
57A0 2807     07650          JR      Z,COMP1
57A2 014000   07660          LD      BC,40H
57A5 09       07670 COMPO    ADD     HL,BC
57A6 3D       07680          DEC     A
57A7 20FC     07690          JR      NZ,COMPO
57A9 C1       07700 COMP1    POP     BC
57AA D1       07710          POP     DE
57AB 7B       07720          LD      A,E
57AC E60F     07730          AND     0FH
57AE C62F     07740          ADD     A,2FH
57B0 85       07750          ADD     A,L
57B1 6F       07760          LD      L,A
```

Program continued

```
57B2 1A       07770          LD      A,(DE)
57B3 FE21     07780          CP      21H             ;ASCII LOW DISPLAY RANGE
57B5 3804     07790          JR      C,COMP3
57B7 FEC0     07800          CP      0C0H            ;ASCII HIGH DISPLAY RANGE
57B9 3802     07810          JR      C,COMP2
57BB 3E2E     07820 COMP3    LD      A,2EH           ;DISPLAY "."/OUT OF RANGE
57BD 77       07830 COMP2    LD      (HL),A
57BE E1       07840          POP     HL
57BF 1C       07850          INC     E
57C0 C3BB56   07860          JP      BLINK
              07870 ;*****************************************************
              07880 ;*              DISPLAY 3 BYTE MESSAGES              *
              07890 ;*****************************************************
57C3 0603     07900 THREE    LD      B,03H           ;3 CHARACTERS LONG
57C5 7E       07910 TO       LD      A,(HL)          ;GET MESSAGE BYTE
57C6 12       07920          LD      (DE),A          ;TO VIDEO
57C7 13       07930          INC     DE              ;BUMP VIDEO
57C8 23       07940          INC     HL              ;BUMP MESSAGE POINTER
57C9 10FA     07950          DJNZ    TO
57CB C9       07960          RET
              07970 ;*****************************************************
              07980 ;*                     MESSAGES                      *
              07990 ;*****************************************************
57CC 43       08000 RETRY    DEFM    'CARE TO RETRY ?'
57DB 03       08010          DEFB    03H
57DC 50       08020 CONT     DEFM    'PRESS "ENTER" TO CONTINUE'
57F5 03       08030          DEFB    03H
57F6 0A       08040 FILESP   DEFB    0AH
57F7 46       08050          DEFM    'FILESPEC ? '
5802 03       08060          DEFB    03H
5803 0A       08070 PSWORD   DEFB    0AH
5804 50       08080          DEFM    'PASSWORD ? '
580F 03       08090          DEFB    03H
5810 52       08100 RIF      DEFM    'RELATIVE SECTOR WITHIN FILE ? '
582E 03       08110          DEFB    03H
582F 52       08120 BESURE   DEFM    'REPLY "Y" TO WRITE SECTOR
5848 0A       08130          DEFB    0AH
5849 C6       08140          DEFB    0C6H
584A 22       08150          DEFM    '"P" TO WRITE SECTOR READ PROTECTED
586C 0A       08160          DEFB    0AH
586D C6       08170          DEFB    0C6H
586E 22       08180          DEFM    '"C" TO WRITE SECTOR NON-READ PROTECTED
5894 0A       08190          DEFB    0AH
5895 C6       08200          DEFB    0C6H
5896 22       08210          DEFM    '"R" TO RELOCATE SECTOR
58AC 0A       08220          DEFB    0AH
58AD C6       08230          DEFB    0C6H
58AE 22       08240          DEFM    '"N" TO ABORT...
58BD 03       08250          DEFB    03H
58BE 0A       08260 DSN      DEFB    0AH
58BF 52       08270          DEFM    'RELATIVE SECTOR NUMBER ? '
58D8 03       08280          DEFB    03H
58D9 44       08290 DN       DEFM    'DRIVE NUMBER ? '
58E8 03       08300          DEFB    03H
58E9 1C1F     08310 MENU     DEFW    1F1CH           ;CONT.CODES (HOME, EREOF)
58EB 2A       08320          DEFM    '*----------------------------------------------------
592B 2A       08330          DEFM    '*--                          DANDYZAP
596B 2A       08340          DEFM    '*--V 1.0-----------------------------------------------
59AB 0A0A     08350          DEFW    0A0AH           ;2 LINE FEEDS
59AD 31       08360          DEFM    '1 - "DISPLAY DISK SECTOR"'
59C6 0A       08370          DEFB    0AH             ;LINE FEED
59C7 32       08380          DEFM    '2 - "DISPLAY FILES SECTOR"'
59E1 0A       08390          DEFB    0AH
59E2 33       08400          DEFM    '3 - "DISPLAY PASSWORD ENCODE"'
59FF 0A       08410          DEFB    0AH
5A00 34       08420          DEFM    '4 - "DISPLAY FILES HASHCODE"'
5A1C 0A       08430          DEFB    0AH
5A1D 35       08440          DEFM    '5 - "RETURN TO TRSDOS"
5A33 0A0A     08450          DEFW    0A0AH
5A35 43       08460          DEFM    'CHOICE ? '
```

```
5A3E 03      08470          DEFB    03H             ;DELIMITER
5A3F 1B1B    08480 CLMESS   DEFW    1B1BH           ;TWO UPWARD LINE FEEDS
5A41 1F03    08490          DEFW    031FH           ;EREOF AND DELIMITER
5A43 1C1F    08500 CLS      DEFW    1F1CH
5A45 0A0A    08510          DEFW    0A0AH
5A47 0A0A    08520          DEFW    0A0AH
5A49 03      08530          DEFB    03H
5A4A 44      08540 DRV      DEFM    'DRV'
5A4D 54      08550 TRK      DEFM    'TRK'
5A50 53      08560 SEC      DEFM    'SEC'
5A53 46      08570 FRS      DEFM    'FRS'
5A56 52      08580 REL      DEFM    'REL'
             08590 ;*****************************************************
             08600 ;* SEARCH MODE FUNCTION  / WRITTEN BY,  JACK WESNIDGE   *
             08610 ;*****************************************************
5A59 3E53    08620 SMODE    LD      A,53H           ; "S"
5A5B CD8C5B  08630          CALL    SET             ;THIS SETS 4020H @ TOP OF
             08640                                  ;SCREEN FOR SIDE DISPLAY
5A5E CDA65B  08650          CALL    SD              ;DISPLAY "F"
5A61 3EBF    08660          LD      A,0BFH
5A63 CDA65B  08670          CALL    SD              ;DISPLAY GRAPHIC BLOCK
5A66 0604    08680          LD      B,04H           ;4 BYTES MAX.
5A68 CD865B  08690          CALL    L1              ;WAS ENTER KEY PRESSED ?
5A6B CA705B  08700          JP      Z,L7            ;RETAIN STRING CONT. FIND
5A6E FE58    08710          CP      58H             ; "X" KEY HIT ?
5A70 CA6357  08720          JP      Z,QUIT          ;IF SO EXIT "S" MODE
5A73 117B5B  08730          LD      DE,FBUF         ;STRING LOCATION
5A76 1805    08740          JR      L8
5A78 CD865B  08750 L3       CALL    L1
5A7B 281C    08760          JR      Z,L2            ;ENTER KEY IS PRESSED
5A7D CDAD56  08770 L8       CALL    ATOHEX          ;REMOVE ASCII BIAS
5A80 301F    08780          JR      NC,BAD          ;IF BAD INPUT
5A82 4F      08790          LD      C,A             ;SAVE FIRST KEY INPUT
5A83 CD865B  08800          CALL    L1              ;SCAN KB
5A86 CDAD56  08810          CALL    ATOHEX          ;REMOVE ASCII BIAS
5A89 3016    08820          JR      NC,BAD          ;IF BAD INPUT
5A8B 6F      08830          LD      L,A             ;SECOND KEY INPUT
5A8C 79      08840          LD      A,C
5A8D CD805B  08850          CALL    AHEX            ;CONVERT ASCII TO HEX
5A90 12      08860          LD      (DE),A          ;STORE IT
5A91 13      08870          INC     DE
5A92 3E8C    08880          LD      A,8CH           ;GRAPHIC BLOCK
5A94 CDA65B  08890          CALL    SD              ;DISPLAY IT
5A97 10DF    08900          DJNZ    L3              ;REPEAT TIL DONE OR ENTER
             08910                                  ;KEY IS PRESSED
5A99 3E04    08920 L2       LD      A,04H           ;MAX. LEN OF STRING
5A9B 90      08930          SUB     B               ;LEFT OVER FROM DJNZ
5A9C 327F5B  08940          LD      (LEN),A         ;STORE LENGTH OF STRING
5A9F 181A    08950          JR      SEARCH
5AA1 CDA65A  08960 BAD      CALL    BAD1
5AA4 180F    08970          JR      B1
5AA6 C5      08980 BAD1     PUSH    BC
5AA7 060F    08990          LD      B,0FH           ;15 LINES ON VIDEO
5AA9 CD8C5B  09000          CALL    SET             ;SETS 4020H @ TOP OF
             09010                                  ;SCREEN FOR SIDE DISPLAY
5AAC 3E20    09020 L6       LD      A,20H           ;SPACE
5AAE CDA65B  09030          CALL    SD              ;DISPLAY @ SIDE OF SCREEN
5AB1 10F9    09040          DJNZ    L6              ;CLEAR SIDE DISPLAY
5AB3 C1      09050          POP     BC
5AB4 C9      09060          RET
5AB5 AF      09070 B1       XOR     A
5AB6 327F5B  09080          LD      (LEN),A         ;ZERO LEN. BAD INPUT
5AB9 189E    09090          JR      SMODE           ;TRY AGAIN...
5ABB 3A1956  09100 SEARCH   LD      A,(MSAVE)       ;GET MODE KEY
5ABE D630    09110          SUB     30H             ;SUB ASCII BIAS
5AC0 3D      09120          DEC     A
5AC1 2802    09130          JR      Z,M1            ;IF MODE 1 THEN JUMP
5AC3 1805    09140          JR      M2              ;JUMP TO MODE 2
5AC5 CDA452  09150 M1       CALL    SK1             ;DISPLAY BUFFER ON SCREEN
5AC8 1803    09160          JR      M3
```

Program continued

```
5ACA CD2F53   09170 M2      CALL    SK2             ;DISPLAY FILE BUFFER
5ACD 2E00     09180 M3      LD      L,00H           ;SECTOR BUFFER
5ACF 2660     09190 NO      LD      H,60H           ;MSB OF READ BUFFER
5AD1 117B5B   09200         LD      DE,FBUF         ;STRING BUFFER
5AD4 1A       09210 AGN     LD      A,(DE)          ;GET CHAR. FROM BUFFER
5AD5 BE       09220         CP      (HL)            ;CP WITH SECTOR BUFFER
5AD6 2855     09230         JR      Z,FOUND
5AD8 2C       09240         INC     L               ;POINT TO NEXT CHAR.
5AD9 20F9     09250         JR      NZ,AGN          ;NO THEN REPEAT
5ADB CDE05A   09260         CALL    SREAD           ;SECTOR READ IF BUF EMPTY
5ADE 18DB     09270         JR      SEARCH
5AE0 D5       09280 SREAD   PUSH    DE
5AE1 E5       09290         PUSH    HL
5AE2 3A1256   09300         LD      A,(PSAVE)       ;GET PROT-STATUS
5AE5 FE06     09310         CP      06H             ;READ-PROTECTED ???
5AE7 2009     09320         JR      NZ,OK           ;IF NOT THEN JUMP
5AE9 ED5BDC55 09330         LD      DE,(AFCB+0AH)   ;CURRENT SECTOR
5AED 13       09340         INC     DE              ;THIS INCREMENTS THE NEXT
5AEE ED53DC55 09350         LD      (AFCB+0AH),DE   ;STORE BACK AS NEW NEXT
5AF2 11D255   09360 OK      LD      DE,AFCB
5AF5 CD3644   09370         CALL    4436H           ;READ A SECTOR
5AF8 F5       09380         PUSH    AF
5AF9 CD2B00   09390         CALL    2BH             ;SCAN KEYBOARD
5AFC B7       09400         OR      A               ;SET FLAG IF KEY PUSHED
5AFD C26357   09410         JP      NZ,QUIT         ;STOP AND DISPLAY SECTOR
5B00 F1       09420         POP     AF
5B01 321256   09430         LD      (PSAVE),A       ;SAVE PROTECTION-STATUS
5B04 E1       09440         POP     HL
5B05 D1       09450         POP     DE
5B06 C8       09460         RET     Z               ;RETURN IF NO ERROR
5B07 FE06     09470         CP      06
5B09 C8       09480         RET     Z               ;RETURN IF ERRORCODE 6
5B0A FE1C     09490         CP      1CH             ;END OF FILE ENCOUNTERED?
5B0C F5       09500         PUSH    AF
5B0D 21435A   09510         LD      HL,CLS
5B10 CD9C55   09520         CALL    DPLAY
5B13 F1       09530         POP     AF
5B14 F5       09540         PUSH    AF
5B15 CC265B   09550         CALL    Z,DONE1
5B18 FE1D     09560         CP      1DH
5B1A CC265B   09570         CALL    Z,DONE1
5B1D F1       09580         POP     AF
5B1E F6C0     09590         OR      0C0H
5B20 CD0944   09600         CALL    4409H           ;DISPLAY ERROR
5B23 C3C15B   09610         JP      RELO            ;EXIT SEARCH MODE
5B26 21935B   09620 DONE1   LD      HL,MES1
5B29 CD9C55   09630         CALL    DPLAY
5B2C C9       09640         RET
5B2D 221756   09650 FOUND   LD      (RBSAVE),HL     ;SAVE POSITION OF FIND
5B30 3A7F5B   09660         LD      A,(LEN)         ;LENGTH OF STRING
5B33 FE01     09670         CP      01H
5B35 282C     09680         JR      Z,OK1           ;IF ONLY ONE BYTE JUMP
5B37 47       09690         LD      B,A             ;LENGTH OF STRING
5B38 05       09700         DEC     B               ;ADJUST FOR FIND...
5B39 2C       09710 NEXT    INC     L               ;POINT TO NXT BUFFER CHAR
5B3A 200C     09720         JR      NZ,SKIP2
5B3C E5       09730         PUSH    HL
5B3D 2ADC55   09740         LD      HL,(AFCB+0AH)   ;GET NEXT VALUE
5B40 2B       09750         DEC     HL
5B41 221556   09760         LD      (NSAVE),HL
5B44 E1       09770         POP     HL
5B45 CCE05A   09780         CALL    Z,SREAD         ;SECTOR READ IF BUF EMPTY
5B48 13       09790 SKIP2   INC     DE              ;BUMP TO NEXT STRING CHAR
5B49 1A       09800         LD      A,(DE)
5B4A BE       09810         CP      (HL)
5B4B 2082     09820         JR      NZ,NO           ;CONT SEARCH WHERE WE
              09830                                 ;LEFT OFF...
5B4D 10EA     09840         DJNZ    NEXT            ;CHECK REST OF STRING
5B4F 7D       09850 SHOW    LD      A,L             ;GET LSB OF BUFFER
5B50 B7       09860         OR      A               ;SET FLAGS
```

```
5B51 2010      09870          JR      NZ,OK1
5B53 2A1556    09880          LD      HL,(NSAVE)      ;GET SAVE NEXT VALUE
5B56 ED5BDC55  09890          LD      DE,(AFCB+0AH)   ;GET CURRENT NEXT VALUE
5B5A DF        09900          RST     18H             ;COMPARE TO EACH OTHER
5B5B 2806      09910          JR      Z,OK1           ;JUMP IF SAME
5B5D 22DC55    09920          LD      (AFCB+0AH),HL   ;ADUSTED NEXT BACK TO FCB
5B60 CDE05A    09930          CALL    SREAD           ;READ IN SECTOR
5B63 2A1756    09940 OK1      LD      HL,(RBSAVE)     ;GET RB OF FIND
5B66 7D        09950          LD      A,L
5B67 5F        09960          LD      E,A
5B68 321456    09970          LD      (ESAVE),A       ;SAVE POSITION IN BUFFER
5B6B 1660      09980          LD      D,60H           ;MSB OF READ BUFFER
5B6D C3BB56    09990          JP      BLINK
5B70 3A1456    10000 L7       LD      A,(ESAVE)       ;GET LSB POSITION OF BUF
5B73 3C        10010          INC     A               ;BUMP TO NEXT POSITION
5B74 6F        10020          LD      L,A
5B75 CCE05A    10030          CALL    Z,SREAD         ;SECTOR READ IF BUF EMPTY
5B78 C3CF5A    10040          JP      NO              ;CONTINUE SEARCH OF OLD
               10050                                  ;STRING...
5B7B 0000      10060 FBUF     DEFW    00H
5B7D 0000      10070          DEFW    00H
5B7F 00        10080 LEN      DEFB    00H
5B80 87        10090 AHEX     ADD     A,A
5B81 87        10100          ADD     A,A
5B82 87        10110          ADD     A,A
5B83 87        10120          ADD     A,A
5B84 85        10130          ADD     A,L
5B85 C9        10140          RET
5B86 CDA35B    10150 L1       CALL    KSD             ;SCAN KYBRD / DISPLAY KEY
5B89 FE0D      10160          CP      0DH
5B8B C9        10170          RET
5B8C 21033C    10180 SET      LD      HL,3C03H        ;THIS SETS SIDE DISPLAY
5B8F 224020    10190          LD      (4020H),HL      ;TO TOP OF SCREEN
5B92 C9        10200          RET
5B93 1F1C      10210 MES1     DEFW    1C1FH
5B95 0A0A      10220          DEFW    0A0AH
5B97 0A        10230          DEFB    0AH
5B98 4E        10240          DEFM    'NO MATCH'
5BA0 0A0A      10250          DEFW    0A0AH
5BA2 03        10260          DEFB    03H
5BA3 CDDC5B    10270 KSD      CALL    KYBRD
5BA6 E5        10280 SD       PUSH    HL
5BA7 2A2040    10290          LD      HL,(4020H)      ;GET CURSOR POSITION
5BAA FE0D      10300          CP      0DH             ;IS IT "ENTER"
5BAC F5        10310          PUSH    AF
5BAD CCA65A    10320          CALL    Z,BAD1
5BB0 F1        10330          POP     AF
5BB1 FE0D      10340          CP      0DH
5BB3 2801      10350          JR      Z,SKIP1
5BB5 77        10360          LD      (HL),A          ;DISPLAY BYTE IN A
5BB6 C5        10370 SKIP1    PUSH    BC
5BB7 014000    10380          LD      BC,40H          ;ONE LINE DOWN
5BBA 09        10390          ADD     HL,BC
5BBB C1        10400          POP     BC
5BBC 224020    10410          LD      (4020H),HL      ;SAVE NEW CURSOR POSITION
5BBF E1        10420          POP     HL
5BC0 C9        10430          RET
5BC1 3A1956    10440 RELO     LD      A,(MSAVE)       ;WHICH MODE ARE WE IN ???
5BC4 D630      10450          SUB     30H             ;SUB ASCII BIAS
5BC6 3D        10460          DEC     A
5BC7 2803      10470          JR      Z,MODE1
5BC9 C37F53    10480          JP      SKIP5
5BCC 21DC57    10490 MODE1    LD      HL,CONT
5BCF CD9C55    10500          CALL    DPLAY           ;DISPLAY "CONTINUE" MESS
5BD2 CDDC5B    10510 K1       CALL    KYBRD           ;SCAN KEYBOARD
5BD5 FE0D      10520          CP      0DH             ;ENTER KEY ???
5BD7 20F9      10530          JR      NZ,K1           ;RESCAN IF NOT...
5BD9 C30052    10540          JP      START           ;BACK TO MAIN MENU
5BDC D5        10550 KYBRD    PUSH    DE
5BDD CD4900    10560          CALL    49H
```

Program continued

```
5BE0 D1        10570          POP     DE
5BE1 C9        10580          RET
5200           10590          END     START
00000 TOTAL ERRORS
```

Slow Scroll

by Peter A. Lewis

Having converted from Level I to Level II, I am extremely pleased with all the new features of the more advanced language. One wrinkle that I don't appreciate, however, is the way the program flies by on the screen when you list it. In this regard, Level I has a superior system by stopping the list when the screen is almost full and allowing you to hit the up arrow key to move the display up the screen.

I know you can press SHIFT@ to freeze the display, but I am invariably fumble-fingered and find that the part of the program that I wanted to see has somehow whisked by before I could stop it. Another thing that bothers me about automatic scrolling is that whenever I write a program that displays more than one screen full of data, I need a PRESS ENTER TO CONTINUE routine to stop the display.

There are three things I don't like about that procedure. First, you have to keep track of the lines that your program is displaying so you know where to insert the pauses. Second, you cannot use the bottom line of the screen because it is needed for the PRESS ENTER. . . message. Third, if the user has the option to output to the printer, your program has to bypass the pause messages.

A small modification to the Level II video driver solves all of the above. With this modification installed, any line that ends with a new line character (ASCII 13) that also causes the screen to scroll, now has the following effect:

- The display freezes after that line is printed.
- Pressing the up arrow allows normal printing to continue until the next new line character (just like Level I LIST).
- Pressing CLEAR clears the screen and starts a new screen.
- Pressing BREAK stops the program or the list and the READY message is displayed.
- Any other key is ignored.

The modification is not active when the cursor is turned on. This allows the screen to scroll normally when you are inputting a long program. You can also use this feature to temporarily turn off the modification within a program. By executing a PRINT CHR$(14) the cursor is turned on and normal scrolling is in effect until you turn it off with a PRINT CHR$(15). The cursor is automatically turned off after an INPUT statement or if the program is restarted from READY.

Three Methods

You can load the modification in three ways. With any method, the program is completely relocatable. My version ends at the top of a 16K machine. You may want to load it at a different location to accommodate a larger memory size or other machine-language routines.

If you have the Editor/Assembler, enter the source code shown in Program Listing 1, assemble it, and create a SYSTEM tape. Then initialize the system by entering SYSTEM followed by /0 and set memory size to 32678. Load the SYSTEM tape and enter a / to execute it. Enter CLEAR, and you're in business. If you want the program to be loaded at a different location in memory, just change the ORG statement. Remember to set memory size to one less than that address.

If you have T-BUG, you can enter the object code shown on the left side of Program Listing 1 by using the M command. Then use the P command to create a SYSTEM tape.

The third method of loading is shown in the BASIC program in Program Listing 2. This routine allows you to specify the load address. (Entering 0 loads the program at the top of 16K.) Don't forget to set MEMORY SIZE first, as described above.

The modification can be turned on or off by executing a PRINT CHR$(1) or PRINT CHR$(0), respectively. Initially it is off.

Program Listing 1. *Source code*

```
                00100 ;SCREEN CONTROL -11/05/79- PETER A. LEWIS
                00120 ;
                00140 ;SET UP DCB DRIVER ADDRESS
                00160 ;
7FA7            00180          ORG     7FA7H           ;32679 DEC.
7FA7 CD0B00     00200 INIT     CALL    000BH           ;PUT LOCN IN HL
7FAA 110A00     00220          LD      DE,SCREEN-$
7FAD 19         00240          ADD     HL,DE
7FAE 221E40     00260          LD      (401EH),HL      ;ENTRY ADDR TO DCB
7FB1 C3CC06     00280          JP      06CCH           ;RETURN TO READY
                00300 ;
                00320 ;CHECK FOR FLAG CHARACTER (00=OFF, 01=ON)
                00340 ;
7FB4 F5         00360 SCREEN   PUSH    AF              ;SAVE FLAGS
7FB5 CD0B00     00380          CALL    000BH           ;PUT LOCN IN HL
7FB8 1801       00400          JR      BYPFLG          ;BYPASS FLAG
7FBA 00         00420          DEFB    0               ;ON/OFF FLAG
7FBB 23         00440 BYPFLG   INC     HL
7FBC 23         00460          INC     HL              ;HL POINTS TO FLAG
7FBD 79         00480          LD      A,C             ;CHARACTER TO A
7FBE E6FE       00500          AND     0FEH            ;LOW BIT OFF
7FC0 2003       00520          JR      NZ,NOFLAG       ;NOT A FLAG
                00540 ;
                00560 ;SAVE NEW FLAG
                00580 ;
7FC2 71         00600          LD      (HL),C          ;STORE NEW FLAG
7FC3 182D       00620          JR      BYPDRV          ;BYPASS DRIVER
                00640 ;
                00660 ;TEST FLAG
                00680 ;
7FC5 7E         00700 NOFLAG   LD      A,(HL)          ;FLAG TO A
7FC6 B7         00720          OR      A               ;TEST FOR ZERO
7FC7 282B       00740          JR      Z,RSTRA         ;0-BYPASS ROUTINE
                00760 ;
                00780 ;TRAP NEW LINE CHARACTER
                00800 ;
7FC9 3A2240     00820          LD      A,(4022H)       ;CURSOR CHAR.
7FCC B7         00840          OR      A               ;IS CURSOR ON?
7FCD 2025       00860          JR      NZ,RSTRA        ;YES-GOTO DRIVER
7FCF 79         00880          LD      A,C             ;CHARACTER TO A
7FD0 FE0D       00900          CP      0DH             ;NEW LINE?
7FD2 2020       00920          JR      NZ,RSTRA        ;NO-TO DRIVER
7FD4 2A2040     00940          LD      HL,(4020H)      ;CURSOR ADDR.
7FD7 114000     00960          LD      DE,40H          ;LINE SIZE
7FDA 19         00980          ADD     HL,DE           ;ADD 1 LINE
7FDB 7C         01000          LD      A,H             ;NEW MSB
7FDC FE40       01020          CP      40H             ;SCREN OVERFLOW?
7FDE 2014       01040          JR      NZ,RSTRA        ;NO-TO DRIVER
                01060 ;
                01080 ;WAIT FOR KEYBOARD ENTRY
                01100 ;
7FE0 CD4900     01120 KBDIN    CALL    49H             ;WAIT FOR KEYBD
7FE3 FE01       01140          CP      01              ;BREAK?
7FE5 2011       01160          JR      NZ,READY        ;YES-SEND READY
7FE7 FE5B       01180          CP      5BH             ;UP ARROW
7FE9 2009       01200          JR      NZ,RSTRA        ;YES-TO DRIVER
7FEB FE1F       01220          CP      1FH             ;CLEAR?
7FED 20F1       01240          JR      NZ,KBDIN        ;NO-READ KEYBD AGAIN
                01260 ;
                01280 ;PROCESS CLEAR KEY
                01300 ;
7FEF CDC901     01320          CALL    01C9H           ;CLS
                01340 ;
                01360 ;BYPASS DRIVER
                01380 ;
7FF2 F1         01400 BYPDRV   POP     AF              ;RESTORE FLAGS
7FF3 C9         01420          RET                     ;BYPASS DRIVER
```

Program continued

```
                01440 ;
                01460 ;RETURN TO DRIVER
                01480 ;
7FF4  F1        01500 RSTRA   POP     AF              ;RESTORE FLAGS
7FF5  C35804    01520         JP      0458H           ;TO DRIVER
                01540 ;
                01560 ;RETURN TO READY
                01580 ;
7FF8  3E0E      01600 READY   LD      A,0EH           ;TURN ON CURSOR
7FFA  322240    01620         LD      (4022H),A
7FFD  C3191A    01640         JP      1A19H           ;TO READY MSG
7FA7            01660         END     INIT
00000 TOTAL ERRORS

BYPDRV 7FF2 01400   00620
BYPFLG 7FBB 00440   00400
INIT   7FA7 00200   01660
KBDIN  7FE0 01120   01240
NOFLAG 7FC5 00700   00520
READY  7FF8 01600   01160
RSTRA  7FF4 01500   00740 00860 00920 01040 01200
SCREEN 7FB4 00360   00220
```

Program Listing 2. *BASIC*

```
 100 REM -SCREEN CONTROL-11/8/79-PETER A. LEWIS
 120 DEFINT B - Z
 140 CLS :
     INPUT "ENTER STARTING ADDRESS FOR LOAD";A
 160 IF A = 0
      THEN
       A = 32679
 170 IF A > 32767
      THEN
       Z = A - 65536 :
      ELSE
       Z = A
 180 FOR X = 0 TO 88
 200  READ D
 220  POKE Z + X,D
 240  NEXT X
 260 PRINT "TO ACTIVATE, ENTER:"
 280 PRINT :
     PRINT ,"SYSTEM":
     PRINT ,"/";A:
     PRINT ,"CLEAR":
     PRINT
 300 PRINT "PRINT CHR$(1) TO TURN ON, PRINT CHR$(0) TO TURN OFF"
 320 PRINT
 340 END
1000 DATA 205,11,0,17,10,0,25,34,30,64,195,25,26,245
1020 DATA 205,11,0,24,1,0,35,35,121,230,254,32,3,113
1040 DATA 24,45,126,183,40,43,58,34,64,183,32,37,121
1060 DATA 254,13,32,32,42,32,64,17,64,0,25,124,254,64
1080 DATA 32,20,205,73,0,254,1,40,17,254,91,40,9,254
1100 DATA 31,32,241,205,201,1,241,201,241,195,88,4,62
1120 DATA 14,50,34,64,195,25,26
```

APPENDIX

Appendix A
Appendix B

APPENDIX A

BASIC Program Listings

Debugging someone else's mistakes is no fun. In a business environment, where programs are continuously updated and programmers come and go, well-commented and structured programs are a must. Indeed, it behooves any serious programmer to learn structured technique.

The BASIC language has no inherent structure. Most interpreters allow remark lines and some are capable of ignoring unnecessary spacing, but BASIC is still more "Beginner's Instruction Code" than "All-purpose."

The listings in this encyclopedia are an attempt at formatting the TRS-80 BASICs. We think it makes them easier to read, easier to trace, and less imposing when it comes time to type them into the computer. You should *not*, however, type them in exactly as they appear. Follow normal syntax and entry procedures as described in your user's manual.

Level I Programs

Programs originally in Level I have been converted to allow running in Level II. To run in Level I, follow this procedure:

- Delete any dimension statements. Example: DIM A (25).
- Change PRINT@ to PRINTAT.
- Make sure that no INPUT variable is a STRING variable.
  Example: INPUT A$ would be changed to INPUT A and subsequent code made to agree.
- Abbreviate all BASIC statements as allowed by Level I.
  Example: *PRINT* is abbreviated *P.*

Model III Users

For the Model I, OUT255,0 and OUT255,4 turn the cassette motor off and on, respectively. For the Model III, change these statements to OUT236,0 and OUT236,2.

APPENDIX B

Glossary

A

access time—the elapsed time between a request for data and the appearance of valid data on the output pins of a memory chip. Usually 200–450 nanoseconds for TRS-80 RAM.

accumulator—the main register(s) in a microprocessor used for arithmetic, shifting, logical, and other operations.

accuracy—generally, the quality or freedom from mistake or error; the extent to which the results of a calculation or a measurement approach the true value of the actual quantities.

acoustic coupler—a connection to a modem allowing signals to be transmitted through a regular telephone handset.

A/D converter—analog to digital converter. See D/A converter.

address—a code that specifies a register, memory location, or other data source or destination.

ALGOL—an acronym for ALGOrithmic Language. A very high-level language used in scientific applications, generally on large-scale computers.

algorithm—a predetermined process for the solution of a problem or completion of a task in a finite number of steps.

alignment—the process of adjusting components of a system for proper interrelationships, including adjustments and synchronization for the components in a system. For the TRS-80, this usually applies to cassette heads and disk drives.

alphanumerics—refer to the letters of the alphabet and digits of the number system, specifically omitting the characters of punctuation and syntax.

alternating current—ac. Electric current that reverses direction periodically, usually many times per second.

ALU—Arithmetic Logic Unit.

analog—the representation of a physical variable by another variable insofar as the proportional relationships are the same over some specified range.

AND—a Boolean logic function. Two operators are tested and, if both are true, the answer is true. Truth is indicated by a high bit, or 1 in machine language, or a positive value in BASIC. If the operators are bytes or words, each element is tested separately. A bit-by-bit logical operation which produces a one in the result bit only if both operand bits are ones.

anode—in a semiconductor diode, the terminal toward which electrons flow from an external circuit; the positive terminal.

APL—A Programming Language; a popular and powerful high-level mathematical language with extensive symbol manipulation.

argument—any of the independent variables accompanying a command.

Arithmetic Logic Unit—ALU. The section of a microprocessor which performs arithmetic functions such as addition or subtraction and logic functions such as ANDing.

array—a collection of data items arranged in a meaningful pattern such as rows and columns which allow the collection and retrieval of data.

ASCII—American Standard Code for Information Interchange. An almost universally accepted code (at least for punctuation and capital letters) where characters and printer commands are represented by numbers between 0 and 255 (base 10). The number is referred to as an ASCII code.

assembler—software that translates operational codes into their binary equivalents on a statement-for-statement basis.

assembly language—a symbolic computer language that is translated by an assembler program into machine language, the numeric codes that are equivalent to microprocessor instructions.

B

backup—1) refers to making copies of all software and data stored externally; 2) having duplicate hardware available.

base—the starting point for representation of a number in written form, where numbers are expressed as multiples of powers of the base value.

BASIC—an acronym for Beginner's All-purpose Symbolic Instruction Code. Developed at Dartmouth College and similar to FORTRAN. The standard, high-level, interactive language for microcomputers.

batch processing—a method of computing in which many of the same types of jobs or programs are done in one machine run. For example, a programming class may type programs on data cards and turn them over to the computer operator. All the cards are put into the card reader, and the results of each person's program are returned later. This is contrasted with interactive computing.

baud—1) a unit of data transmission speed equal to the number of code elements (bits) per second; 2) a unit of signaling speed equal to the number of discrete conditions or signal events per second.

baud rate—a measure of the speed at which serial data is transmitted electronically. The equivalent of bits per second (bps) in microcomputing.

benchmark—to test performance against a known standard.

BCD—binary coded decimal. The 4-bit binary notation in which individual decimal digits (0 through 9) are represented by 4-bit binary numerals; e.g., the number 23 is represented by 0010 0011 in the BCD notation.

bias—a dc voltage applied to a transistor control electrode to establish the desired operating point.

bidirectional bus—a bus structure used for the two-way transmission of signals, that is, both input and output.

bidirectional printer—a printer capable of printing both left-to-right and right-to-left. Data is prestored in a fixed-size buffer.

binary—a number system which uses only 0 and 1 as digits. It is the equivalent of base 2. Used in microcomputing because it is easy to represent 1s and 0s by high and low electrical signals.

binary digit—the two digits, 0 and 1, used in binary notation. Often shortened to bit.

bi-stable—two-state

bit—an abbreviation for binary digit. A 0 or 1 in the binary number system. A single high or low signal in a computer.

bit position—the position of a binary digit within a byte or larger group of binary digits. Bit positions in the Model I, II, III, and Color Computer are numbered from right to left, zero through N. This number corresponds to the power of two represented.

Boolean algebra—a mathematical system of logic first identified by George Boole, a 19th century English mathematician. Routines are described by combinations of ANDs, ORs, XORs, NOTs, and IF-THENs. All computer functions are based upon these operators.

boot—short for bootstrap loader or the use of one. The bootstrap loader is a very short routine whose purpose is to load a more sophisticated system into the computer when it is first turned on. On some machines it is keyed in, and on others it is in read only memory (ROM). Using this program is called booting or cold-starting the system.

bps—bits per second.

buffer—memory set aside temporarily for use by the program. Particularly refers to memory used to make up differences in the data transfer rates of the computer and external devices such as printers and disks.

bug—an error in software or hardware.

bus—an ordered collection of all address, data, timing, and status lines in the computer.

byte—eight bits that are read simultaneously as a single code.

C

CAI—an acronym for Computer Aided Instruction.

card—a specially designed sheet of cardboard with holes punched in specific columns. The placement of the holes represents machine-readable data. Also a term referring to a printed circuit board.

card reader—a device for reading information from punched cards.

cassette recorder—a magnetic tape recording and playback device for entering or storing programs.

cathode—in a semiconductor diode, the terminal from which electrons flow to an external circuit; the negative terminal.

character—a single symbol that is represented inside the computer by a specific code.

checksum—a method of detecting errors in a block of data by adding each piece of data in the block to a sum and comparing the final result to a predetermined result for the block of data.

chip—the shaped and processed semiconductor die mounted on a substrate to form a transistor or other semiconductor device.

circuit—a conductor or system of conductors through which an electric current may flow.

circuit card—a printed circuit board containing electronic components.

clear—to return a memory to a non-programmed state, usually represented as 0 or OFF (empty).

clock—a simple circuit that generates the synchronization signals for the microprocessor. The speed or frequency of this clock directly affects the speed at which the computer can perform, regardless of the speed of which the individual chips are capable.

COBOL—COmmon Business-Oriented Language. A language used primarily for data processing. Allows programming statements that are very similar to English sentences.

compiler—software that will convert a program written in a high-level language to binary code, on a many-for-one basis.

complement—a mathematical calculation. In computers it specifically refers to inverting a binary number. Any 1 is replaced by a 0, and vice versa.

computer interface—a device designed for data communication between a central computer and another unit such as a programmable controller processor.

concatenate—to put two things, each complete by itself, together to make a larger complete thing. In computers this refers to strings of characters or programs.

conductor—a substance, body, or other medium that is suitable to carry an electric current.

constant—a value that doesn't change.

CPU—central processing unit. The circuitry that actually performs the functions of the instruction set.

CRT—cathode ray tube. In computing this is just the screen the data appears on. A TV has a CRT.

cue—refers to positioning the tape on a cassette unit so that it is set up to a read/write section of tape.

cursor—a visual movable pointer used on a CRT by the programmer to indicate where an instruction is to be added to the program. The cursor is also used during editing functions.

cycle—a specific period of time, marked in the computer by the clock.

D

D/A converter—digital to analog converter. Common in interfacing computers to the outside world.

daisy wheel—a printer type which has a splined character wheel.

data—general term for numbers, letters, symbols, and analog quantities that serve as information for computer processing.

data base—refers to a series of programs each having a different function, but all using the same data. The data is stored in one location or file and each program uses it in a fashion that still allows the other program to use it.

data entry—the practice of entering data into the computer or onto a storage device. Knowledge of operating or programming a computer is not necessary for a data entry operator.

debug—to remove bugs from a program.

decrement—to decrease the value of a number. In computers the number is in memory or a register, and the amount it is decremented is usually one.

dedicated—in computer terminology, a system set up to perform a single task.

default—that which is assumed if no specific information is given.

degauss—to demagnetize. Must be done periodically to tape and disk heads for reliable data transfer.

diagnostic program—a test program to help isolate hardware malfunctions in the programmable controller and application equipment.

digital—the representation of data in binary code. In microcomputers, a high electrical signal is a 1 and a low signal is a 0.

digital circuit—an electronic network designed to respond at input voltages at one level, and similarly, to produce output voltages at one level.

diode—a device with an anode and a cathode which permits current flow in one direction and inhibits current flow in the other direction.

direct current—dc. Electric current which flows in only one direction; the term designates a practically non-pulsating current.

disassembly—remaking an assembly source program from a machine-code program.

disk—an oxide-coated, circular, flat object, in a variety of sizes and containers, on which computer data can be stored.

disk controller—an interface between the computer and the disk drive.

disk drive—a piece of hardware that rotates the disk and performs data transfer to and from the disk.

disk operating system—DOS. The system software that manipulates the data to be sent to the disk controller.

dividend—the number that is divided by the divisor. In A/B, A is the dividend.

divisor—the number that "goes into" the dividend in a divide operation. In A/B, B is the divisor.

DMA—direct memory access. A process where the CPU is disabled or

bypassed temporarily and memory is read or written to directly.

documentation—a collection of written instructions necessary to use a piece of hardware, software, or a system.

dot-matrix printer—instead of each letter having a separate type head (like a typewriter), a single print head makes the characters by printing groups of dots. The print is not as easy to read, but such printers are less expensive to manufacture.

downtime—the time when a system is not available for production due to required maintenance.

driver—a small piece of system software used to control an external device such as a keyboard or printer.

dump—to write data from memory to an external storage device.

duplex—refers to two-way communications taking place independently, but simultaneously.

dynamic memory—circuits that require a periodic (every few milliseconds) recharge so that the stored data is not lost.

E

EAROM—an acronym for Electrical Alterable Read Only Memory. The chip can be read at normal speed, but must be written to with a slower process. Once written to, it is used like a ROM, but can be completely erased if necessary.

editor—a program that allows text to be entered into memory. Interactive languages usually have their own editors.

EOF—End Of File.

EOL—End Of Line (of text).

EPROM—Erasable Programmable Read Only Memory. A read only memory in which stored data can be erased by ultraviolet light or other means and reprogrammed bit-by-bit with appropriate voltage pulses.

Exclusive OR—a bit-by-bit logical operation which produces a one bit in the

result only if one or the other (but not both) operand bits is a one.

execution—the performance of a specific operation such as would be accomplished through processing one instruction, a series of instructions, or a complete program.

execution cycle—a cycle during which a single instruction of one specific operation is performed.

execution time—the total time required for the execution to actually occur.

expansion interface—a device attached to the computer that allows a greater amount of memory or attachment of other peripherals.

exponent—the power to which a floating-point number is raised.

F

fetch cycle—a cycle during which the next instruction to be performed is read from memory.

field-effect transistor—FET. A transistor in which the resistance of the current path from the source to drain is modulated by applying a transverse electric field between grid or gate electrodes; the electric field varies the thickness of depletion layers between the gates, thereby reducing the conductance.

file—a set of data, specifically arranged, that is treated as a single entity by the software or storage device.

firmware—software that is made semi-permanent by putting it into some type of ROM.

flag—a single bit that is high (set) or low (reset), used to indicate whether or not certain conditions exist or have occurred.

flip-flop—a bi-stable device that assumes either of two possible states such that the transition between the states must be accomplished by electronic switching.

floating-point number—a standard way of representing any size number in computers. Floating-point numbers contain a fractional portion (mantissa) and power of two (exponent) in a form similar to scientific notation.

flowcharting—a method of graphically displaying program steps, used to develop and define an algorithm before writing the actual code.

FORTRAN—FORmula TRANslator. One of the first high-level languages, written specifically to allow easy entry of mathematical problems.

full duplex—a mode of data transmission that is the equivalent of two paths—one in each direction simultaneously.

G

game theory—see von Neumann.

garbage—computer term for useless data.

gate—a circuit that performs a single Boolean function. A circuit having an output and a multiplicity of inputs, so designed that the output is energized only when a certain combination of pulses is present at the inputs.

GIGO—Garbage In, Garbage Out. One of the rules of computing. If the data going into the computer is bad, the data coming out will be bad also.

graphics—information displayed pictorially as opposed to alphanumerically.

ground—a conducting path, intentional or accidental, between an electric circuit or equipment and the earth, or some conducting body serving in place of the earth.

H

H—a suffix for hexadecimal, e.g., 4FFFH.

half duplex—data can flow in both directions, but not simultaneously. See duplex.

handshaking—a term used in data transfer. Indicates that beside the data lines there are also signal lines so both devices know precisely when to send or receive data. Handshaking requires clocking pulses on both ends of the communications line. Contrast with buffer.

hard copy—a printout; any form of printed document such as a ladder diagram, program listing, paper tape, or punched cards.

hardware—refers to any physical piece of equipment in a computer system.

hex—hexadecimal.

hexadecimal—representation of numbers in base sixteen by use of the hexadecimal digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F.

high—a signal line logic level. The computer senses this level and treats it as a binary 1.

high-level language—a programming language which is CPU-independent and closely resembles English.

high order—see most significant bit.

HIT—acronym for Hash Index Table. A section of the directory on a TRS-80 disk.

human engineering—usually refers to designing hardware and software with ease of use in mind.

I

IC—integrated circuit.

immediate—addressing mode in which the address of the information that an operation is supposed to act upon immediately follows the operation code.

increment—to increase, usually by one. See decrement.

indexed—addressing mode where the information is addressed by a specified value, or by the value in a specified register.

indirect—addressing mode in which the address given points to another address, and the second address is where the information actually is.

input devices—devices such as limit switches, pressure switches, push buttons, etc., that supply data to a programmable controller. These discrete inputs are two types: those with common return, and those with individual returns (referred to as isolated inputs). Other inputs include analog devices and digital encoders.

instruction—a command or order that will cause a computer to perform one particular operation.

integer variable—a BASIC variable type. It can hold values of −32,768 through +32,767 in two-byte two's complement notation.

integrated circuit—IC. An interconnected array of active and passive elements integrated with a single semiconductor substrate or deposited on the substrate and capable of performing at least one electronic circuit function. See chip.

intelligent terminal—a terminal with a CPU and a certain amount of memory that can organize the data it receives and thus achieve a high level of handshaking with the host computer.

interactive computing—refers to the appearance of a one-to-one human-computer relationship.

interface—a piece of hardware, specifically designed to hook two other devices together. Usually some software is also required.

interpreter—a piece of system software that executes a program written in a high-level language directly. While useful for interactive computing, this system is too slow for most serious programming. Contrast with compiler.

interrupt—a signal that tells the CPU that a task must be done immediately. The registers are pushed to the stack, and a routine for the interrupt is branched to. When finished, the registers are popped from the stack and the main program continues.

I/O—acronym for input/output. Refers to the transfer of data.

iteration—one pass through a given set of instructions.

J

jack—a socket, usually mounted on a device, which will receive a plug (generally mounted on a wire).

K

K—abbreviation for kilo. In computer terms 1024, in loose terms 1000.

L

language—a set of symbols and rules for representing and communicating information (data) among people, or between people and machines.

large scale integration—LSI. Any integrated circuit which has more than 100 equivalent gates manufactured simultaneously on a single slice of semiconductor material.

least significant bit—the rightmost bit in a binary value, representing 2^0.

least significant byte—refers to the lowest position digit of a number. The rightmost byte of a number or character string.

LIFO—acronym for Last In First Out. Most CPUs maintain a "stack" of memory. The last data pushed onto the stack is the first popped out.

light emitting diode—LED. A semiconductor diode that displays alphanumeric characters when supplied with a specified voltage.

light pen—a device that senses light, interfaced to the computer for the purpose of drawing on the CRT screen.

line—in communications, describes cables, telephone lines, etc., over which data is transmitted to and received from the terminal.

line printer—a high-speed printing device that prints an entire line at one time.

location—a storage position in memory.

logic—a means of solving complex problems through the repeated use of simple functions which define basic concepts. Three basic logic functions are AND, OR, and NOT.

logic diagram—a drawing which represents the logic functions AND, OR, NOT, etc.

logic level—the voltage magnitude associated with signal pulses representing ones and zeroes (1s and 0s) in binary computation.

logical shift—a type of shift in which an operand is shifted right or left, with a zero filling the vacated bit position.

loop—a set of instructions that executes itself continuously. If the programmer has the presence of mind to provide for a test, the loop is discontinued when the test is met, otherwise it goes on until the machine is shut down.

loop counter—one way to test a loop. The counter is incremented at each pass through the loop. When it reaches a certain value, the loop is terminated.

low—a logic signal voltage. The computer senses this as a binary 0.

lsb—see least significant bit.

LSI—acronym for Large Scale Integration. An integrated circuit with a large number of circuits such as a CPU. See chip.

M

machine code—refers to programming instructions that are stored in binary and can be executed directly by the CPU without any compilation, interpretation, or assembly.

machine language—the primary instructions that were designed into the CPU by the manufacturer. These instructions move data between memory and registers, perform simple adding in registers, and allow branching based on values in registers.

macro—a routine that can be separately programmed, given a name, and executed from another program. The macro can perform functions on variables in the program that called it without disturbing anything else and then return control to the calling program.

mainframe—refers to the CPU of a computer. This term is usually confined to larger computers.

mantissa—the fractional portion of a floating-point number.

matrix—a two-dimensional array of circuit elements, such as wires, diodes, etc., which can transform a digital code from one type to another.

memory—the hardware that stores data for use by the CPU. Each piece of

data (bit) is represented by some type of electrical charge. Memory can be anything from tiny magnetic doughnuts to bubbles in a fluid. Most microcomputers have chips that contain many microscopic capacitors, each capable of storing a tiny electrical charge.

metal oxide semiconductor—MOS. A metal insulator semiconductor structure in which the insulating layer is an oxide of the substrate material; for a silicon substrate the insulator is silicon oxide.

micro electronics—refers to circuits built from miniaturized components and includes integrated circuits.

microprocessor—an electronic computer processor section implemented in relatively few IC chips (typically LSI) which contain arithmetic, logic, register, control, and memory functions.

microsecond—us. One millionth of a second: 1×10^{-6} or 0.000001 second.

millisecond—ms. One thousandth of a second: 10^{-3} or 0.001 second.

minuend—the number from which the subtrahend is subtracted.

mixed number—a number consisting of an integer and fraction as, for example, 4.35 or (binary) 1010.1011.

mnemonic—a short, alphanumeric abbreviation used to represent a machine-language code. An assembler will take a program written in these mnemonics and convert it to machine code.

modem—MOdulator/DEModulator. An I/O device that allows communication over telephone lines.

module—an interchangeable plug-in item containing electronic components which may be combined with other interchangeable items to form a complete unit.

monitor—1) a CRT; 2) a short program that displays the contents of registers and memory locations and allows them to be changed. Monitors can also allow another program to execute one instruction at a time, saving programs and disassembling them.

MOS—see metal oxide semiconductor.

MOSFET—metal oxide semiconductor field effect transistor.

most significant bit—the leftmost bit in a binary value, representing the highest-order power of two. In two's complement notation, this bit is the sign bit.

most significant byte—the highest-order byte. In the multiple-precision number A13EF122H, A1H is the most significant byte.

msb—see most significant byte.

multiplexing—a method allowing several sets of data to be sent at different times over the same communication lines, yet all of the data can be used simultaneously after the final set is received. For example, several LED displays, each requiring four data lines, can all be written to with only one group of four data lines. The same concept is used with communication lines.

multiplicand—the number to be multiplied by the multiplier.

multiplier—the number that is multiplied against the multiplicand. The number "on the bottom."

N

NAND—an acronym for NOT AND. A Boolean logic expression. AND is performed, then NOT is performed to the result.

nanosecond—one billionth of a second.

nesting—putting one loop inside another. Some computers limit the number of loops that can be nested.

noise—extraneous signals; any disturbance which causes interference with the desired signal or operation.

non-volatile memory—a memory that does not lose its information while its power supply is turned off.

NOT—a Boolean operator that reverses outputs (1 becomes 0, 0 becomes 1). This is the one's complement.

O

object code—all of the machine code that is generated by a compiler or assembler. Once object code is loaded into memory it is called machine code.

octal—refers to the base 8 number system, using digits 0–7.

OEM—Original Equipment Manufacturer.

off-line—describes equipment or devices which are not connected to the communications line.

off-the-shelf—a term referring to software. A generalized program that can be used by many computer owners. It is mass produced and can be bought off-the-shelf.

on-line—a term describing a situation where one computer is connected to another, with full handshake, over a modem line.

operands—the numeric values used in the add, subtract, or other operation.

OR—a Boolean logic function. If at least one of the lines tested is high (binary 1), the answer is high.

output—the current, voltage, power, driving force, or information which a circuit or a device delivers. The terminals or other places where a circuit or device can deliver energy.

output devices—devices such as solenoids, motor starters, etc., that receive data from the programmable controller.

overflow—a condition that exists when the result of an add, subtract, or other arithmetic operation is too large to be held in the number of bits allotted.

overlay—a method of decreasing the amount of memory a program uses by allowing sections that are not in use simultaneously to load into the same area of memory. The new routine destroys the first routine, but it can always be loaded again if needed. Usually used in system programs.

oxide—an iron compound coating on tapes and disks that allows them to be magnetized so that they can be read by electrical devices and the information converted back to machine code.

P

page—refers to a 256 (2 to the 8th power) word block of memory. How large a word depends on the computer. Most micros are eight-bit word machines. Many chips do special indexed and offset addressing on the page where the program counter is pointing and/or on the first page of memory.

parallel—describes a method of data transfer where each bit of a word has its own data line, and all are transferred simultaneously. Contrast with serial.

parameter—a variable or constant that can be defined by the user and usually has a default value.

parity—a method of checking accuracy. The parity is found by adding all the bits of a word together. If the answer is even, the parity is 0 or even. If odd, the parity is 1 or odd. The bit sometimes replaces the most significant bit and usually sets a flag.

parity bit—an additional bit added to a memory word to make the sum of the number of 1s in a word always even or odd as required.

parity check—a check that tests whether the number of 1s in an array of binary digits is odd or even.

PC board—see printed circuit board.

peripheral devices—a generic term for equipment attached to a computer, such as keyboards, disk drives, cassette tapes, printers, plotters, speech synthesizers.

permutation—arrangements of things in definite order. Two binary digits have four permutations: 00, 01, 10, and 11.

PILOT—a simple language for handling English sentences and strings of alphanumeric characters. Generally used for CAI.

PL/1—an acronym for Programming Language 1. A programming language used by very large computers. It incorporates most of the better features from other programming languages. Its power comes from the fact that bits can be manipulated from the high-level language.

plotter—a device that can draw graphs and curves and is controlled by the computer through an interface.

port—a single addressable channel used for communications.

positional notation—representation of a number where each digit position represents an increasingly higher power of the base.

precision—the number of significant digits that a variable or number format may contain.

printed circuit board—a piece of plastic board with lines of a conductive material deposited on it to connect the components. The lines act like wires. These can be manufactured quickly and are easy to assemble the components on.

processor—a unit in the programmable controller which scans all the inputs and outputs in a predetermined order. The processor monitors the status of the inputs and outputs in response to the user-programmed instructions in memory, and it energizes or de-energizes outputs as a result of the logical comparisons made through these instructions.

product—the result of a multiply.

program—a sequence of instructions to be executed by the processor to control a machine or process.

PROM—Programmable Read Only Memory. A memory device that is written to once and from then on acts like a ROM.

pseudo code—a mnemonic used by assemblers that is not a command to the CPU, but a command to the assembler itself.

punched-card equipment—peripheral devices that enable punching or reading paper punched cards that hold character or binary data.

Q

quotient—the result of a divide operation.

R

RAM—acronym for Random Access Memory. An addressable LSI device used to store information in microscopic flip-flops or capacitors. Each may be set to an ON or OFF state, representing logical 1 or 0. This type of

memory is volatile, that is to say, memory is lost while power is off, unless battery backup is used.

read—to sense the presence of information in some type of storage, which includes RAM memory, magnetic tape, punched tape, etc.

real time clock—a clock in the sense that we normally think of one, interfaced to the computer.

record—a file is divided into records, each of which is organized in the same manner.

register—a fast-access memory location in the microprocessor. Used for holding intermediate results and for computation in machine language.

relative addressing—an address that is dependent upon where the program counter is presently pointing.

remainder—the amount of dividend remaining after a divide has been completed.

ROM—an acronym for Read Only Memory. Memory that is addressed by the bus, but can only be read from. If you tell the CPU to write to it, the machine will try, but the data is not remembered.

rounding—the process of truncating bits to the right of a bit position and adding zero or one to the next higher bit position based on the value to the right. Rounding the binary fraction 1011.1011 to two fractional bits, for example, results in 1011.11.

RPG—an acronym for Report Program Generator. A language for business that primarily reads data from cards and prints reports containing that data.

RS-232—an interface that converts parallel data to serial data for communications purposes. The output is universally standard.

S

scaling—multiplying a number by a fixed amount so that a fraction can be processed as an integer value.

scientific notation—a standard form for representing any size number by a mantissa and power of ten.

semiconductor—a compound that can be made to vary its resistance to electricity by mixing it differently. Layers of this material can be used to make circuits that do the same things tubes do, but using much less electricity. Transistors and integrated circuits are made from semiconductive material and are called semiconductors.

serial—a way of sending data, one bit at a time, between two devices. The bits are rejoined into bytes by the receiving device. Contrast with parallel.

sign bit—sometimes the most significant bit is used to indicate the sign of the number it represents. 1 is negative (-) and 0 is positive (+).

signed numbers—numbers that may be either positive or negative.

significant bits—the number of bits in a binary value after leading zeros have been removed.

significant digit—a digit that contributes to the precision of a number. The number of significant digits is counted beginning with the digit contributing the most value, called the most significant digit, and ending with one contributing the least value, called the least significant digit.

simulator—a computer that is programmed to mimic the action and functions of another piece of machinery, usually for training purposes. A computer is usually employed because it is cheaper to have the computer simulate these actions than to use the real thing. Airplane and power plant trainers are excellent examples.

software—refers to the programs that can be run on a computer.

solid state devices (semiconductors)—electronic components that control electron flow through solid materials such as crystals; e.g., transistors, diodes, integrated circuits.

source program—the program written in a language or mnemonics that is converted to machine code. The source program as well as the object code generated from it can be saved in mass storage devices.

SPOOL—acronym for Simultaneous Peripheral Output, On-Line. Used to overlap processing, typically, with printing.

stack—an area of memory used by the CPU and the programmer particularly for storage of register values during interrupt routines. See LIFO.

stepper motor—a special motor in a disk drive that moves the read/write head a specific distance each time power is applied. That distance defines the tracks on a disk.

storage—see memory.

subroutine—a routine within a program that ends with an instruction to return program flow to where it was before the routine began. This routine is used many times from many different places in the program, and the subroutine allows you to write the code for that routine only once. Similar to a macro.

subtrahend—the number that is subtracted from the minuend.

syntax—the term is used exactly as it is used in English composition. Every language has its own syntax.

system—a collection of units combined to work as a larger integrated unit having the capabilities of all the separate units.

system software—software that the computer must have loaded and running to work properly.

T

table—an ordered collection of variables and/or values, indexed in such a way that finding a particular one can be done quickly.

tape reader—a unit which is capable of sensing data from punched tape.

Teletype™—a peripheral electromechanical device for entering or outputting a program or data in either a punched paper tape or printed format.

text editor—see word processor.

time sharing—refers to systems which allow several people to use the computer at the same time.

track—a concentric area on a disk where data is stored in microscopic magnetized areas.

transistor—an active component of an electronic circuit consisting of a small

block of semiconducting material to which at least three electrical contacts are made, usually two closely spaced rectifying contacts and one ohmic (non-rectifying) contact; it may be used as an amplifier, detector, or switch.

transistor-transistor logic—TTL. A logic circuit containing two transistors, for driving large output capacitances at high speed. A family of integrated circuit logic. (Usually 5 volts is high or 1, and 0 volts is low or 0; 5V = 1, 0V = 0).

truncation—the process of dropping bits to the right of a bit position. Truncating the binary fraction 1011.1011 to a number with fraction of two bits, for example, results in 1011.10.

truth table—a table defining the results for several different variables and containing all possible states of the variables.

TTL—see transistor-transistor logic.

TTY—an abbreviation for Teletype.

two's complement—a standard way of representing positive and negative numbers in microcomputers.

U

unsigned numbers—numbers that may be only positive; absolute numbers.

utility—a program designed to aid the programmer in developing other software.

UV erasable PROM—an ultraviolet erasable PROM is a programmable read-only memory which can be cleared (set to 0) by exposure to intense ultraviolet light. After being cleared, it may be reprogrammed.

V

variable—a labeled entity that can take on any value.

volatile memory—a memory that loses its information if the power is removed from it.

von Neumann, John (1903–1957)—mathemetician. He put the concept of games, winning strategy, and different types of games into mathematical formulae. He also advanced the concept of storing the program in memory as opposed to having it on tape.

W

weighted value—the numerical value assigned to any single bit as a function of its position in the code word.

word—a grouping or a number of bits in a sequence that is treated as a unit and is stored in one memory location. If the CPU works with 8 bits, then the word length is 8 bits. Common word sizes are 4, 8, 12, 16, and 32. Some are as large as 128 bits.

word processor—a computer system dedicated to editing text and printing it in various controllable formats. See editor.

write—to store in memory or on a mass storage device.

X

XOR—a Boolean function. Acronym for eXclusive OR. Similar to OR but answer is high (1) if and only if one line is high.

Z

zero flag—a bit in the microprocessor used to record the zero/non-zero status of the result of a machine-language instruction.

zero page—refers to the first page of memory.

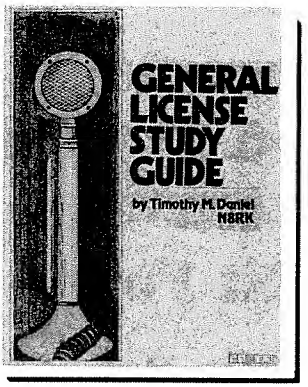
INDEX

INDEX

INDEX COMPILED BY NAN MCCARTHY

Wayne Green Books

GENERAL LICENSE STUDY GUIDE



by Timothy M. Daniel N8RK

Understanding, not memorization, is the learning approach stressed in the *General License Study Guide.* The follow-up to the *Novice License Study Guide,* this revised and up-to-date reference book makes it easy to gain the knowledge needed to earn ham radio's most popular ticket. Rules and regulations, electronics fundamentals, operating procedures, and even tips on taking the exam are included. The key is learning and understanding—the real fun of ham radio. A veritable wealth of diagrams, tables, and charts, the *General License Study Guide* also includes:

- FCC Rules and Regulations
- Study questions and answers for each chapter
- Comprehensive review

ISBN 0-88006-017-4 96 pages **$6.95**

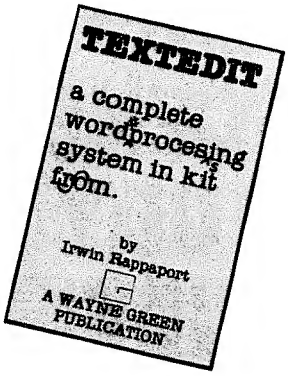
WAYNE GREEN BOOKS
Division of Wayne Green Inc.
Peterborough, NH 03458
FOR TOLL FREE ORDERING:
1-800-258-5473
*TRS-80 is a trademark of Radio Shack division of Tandy Corp.

Wayne Green Books

TEXTEDIT

A Complete Word Processing System in Kit Form

by Irwin Rappaport



Word processing systems can cost hundreds of dollars and, even when you've bought one, it probably won't do everything you want.

TEXTEDIT is an inexpensive word processor that you can adapt to suit your needs, from writing form letters to large texts. It is written in modules, so you can load and use only those portions that you need. Included are modules that perform:

- right justification
- ASCII upper/lowercase conversion
- one-key phrase entering
- complete editorial functions
- and much more!

TEXTEDIT is written in TRS-80* Disk BASIC, and the modules are documented in the author's clear writing style. Not only does Irwin Rappaport explain how to use **TEXTEDIT**; he also explains programming techniques implemented in the system.

TEXTEDIT is an inexpensive word processor that helps you learn about BASIC programming. It is written for TRS-80 Models I and III with TRSDOS 2.2/2.3 and 32K.

ISBN 0-88006-050-6 90 pages **$9.97**

WAYNE GREEN BOOKS
Division of Wayne Green Inc.
Peterborough, NH 03458
FOR TOLL FREE ORDERING:
1-800-258-5473

The real value of your computer lies in your ability to use it. The capabilities of the TRS-80* are incredible if you have the information which will help you get the most from it. Little of this information is available in your instruction books.

The *Encyclopedia for the TRS-80* will teach you how to get the most from your computer. In addition to a wealth of programs which are ready for you to use, reviews of accessories and commercially available programs, you will also learn how to write your own programs or even modify commercial programs for your own specific use.

The *Encyclopedia* is packed with practical information, written and edited for the average TRS-80 owner, not the computer scientist. You will find it interesting and valuable.

Wayne Green
Publisher

$10.95 ISBN 0-88006-047-6